”A model-driven approach for designing multi-platform user interface dialogues”

Mbaki Luzayisu

"A model-driven approach for designing multi-platform user interface dialogues"

Mbaki Luzayisu, Efrem

Abstract

Human-computer interaction becomes sophisticated, multimodal and multi device and needs to be well-designed with the aim of facilitating application correction (i.e. to correcting errors/bugs in the application) or extension (i.e. adding new functionalities or modifying existing tasks). This thesis is focused on building a methodology of designing and specifying User Interface (UI) behaviour. The Unified Modelling Language (UML) is used to describe in detail the conceptual model and to define all its objects. The methodology flux diagram is provided with the specification of the consistency and the completeness properties of the transformation model. To support the methodology, we implement a graphic Dialog Editor in which Models are organized in three levels (abstract, concrete and final) according to Cameleon Reference Framework (CFR) and, whose process respects the Model Driven Engineering (MDE) approach. Furthermore, the use of Dialog Editor is illustrated through a simple exam...

Document type : *Thèse (Dissertation)*

A Model-Driven Approach for Designing Multi-platform User Interface Dialogues

By **Efrem MBAKI LUZAYISU**

A thesis submitted in fulfilment of the requirements for the degree of

Doctor of Computer Science
of the Ecole Polytechnique de Louvain,
Université catholique de Louvain

Examination committee:

Prof. **François BODART**, Facultés Univ. Notre-Dame de la Paix, Reader
Prof. **Manuel KOLP**, Université catholique de Louvain, Reader
Prof. **Christophe KOLSKI**, Université de Valenciennes, France, Examiner
Prof. **Jean VANDERDONCKT**, Université catholique de Louvain, Advisor
Prof. **Christophe De VLEESCHOUWER**, Université catholique de Louvain, President
Prof. **Marco WINCKLER**, Université Paul Sabatier, France, Examiner

March 2013

Acknowledgements

I would like to express my thanks to:

- My advisor, Prof. Jean Vanderdonckt, for his constant support and enthusiasm regarding my work.
- Professors François Bodart, Christophe Kolski, Manuel Kolp, Christophe De Vleeschouwer and Marco Winckler for accepting to participate to the jury of this thesis and for their constructive comments brought to earlier versions of this manuscript.
- My colleagues from the Louvain Interaction Laboratory (Lilab) at Université catholique de Louvain and from Orfival SA.
- My wife (Virginie Bozzo Lelo) and my children (Evy, Cecilia and Lys)
- All my family and friends.

This thesis has been achieved thanks to the support of:

- Pole of Research on Information and Services Management and Engineering (PRISME) at LSM.
- The SIMILAR network of excellence supported by the 6th Framework Program of the European Commission, under contract FP6-IST1-2003-507609 (www.similar.cc).
- The UsiXML Project supported by the ITEA2 Call 3 Program of the European Commission, under contract 2008026 (www.UsiXML.org).
- The SERENOA Project supported by the 7th Framework Program of the European Commission, under contract FP7-ICT5-258030 (www.serenoa-fp7.eu)

Table of Contents

Table of Figures

Table of tables

Table of Abbreviations

ABBREVIATION	FULL NAME
AC	Abstract Container
ADO	ActiveX Data Object
AGG	Attributed Graph Grammar
AIC	Abstract Individual Components
AIO	Abstract Interaction Objects
AUI	Abstract User Interface
BNF	Backus-Naur Form
CIM	Computing-Independent Model
CIO	Concrete Interaction Objects
CRF	Cameleon Reference Framework
CSS	Cascading Style Sheets
CTI	Congo Transfer International
CTT	ConcurTaskTree
CUI	Concrete User Interface
CUI	Concrete User Interface
CUSQ	Computer Usability Satisfaction Questionnaires
DBMS	Data Base Management Systems
DSL	Domain-Specific Language
DSM	Domain-Specific Model
ECA	Event-Condition-Action
FUI	Final User Interface
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HTA	HTML for Applications
HTML	HyperText Markup Language
IDE	Integrated Development Environment
InfoQual	Information Quality
IntQual	Interface Quality
IOG	Interaction Object Graph
IOG	Interactive Objects
IS	Information System
KBE	Knowledge-Based Engineering
M2C	Model-to-Code
M2M	Model-to-Model
MDA	Model-Driven Architecture

MDE	Model-Driven Engineering
MIPIM	Multimodal Interface Presentation and Interaction Model
PC	Personal Computer
PSM	Platform-Specific Model
SysUse	System Usefulness
TA	Transformation Atom
TAG	Task-Action Grammar
TM	Transformation Mapping
TR	Transformation Rule
UI	User Interface
UIDL	User Interface Description Language
UIML	User Interface Markup Language
USIXML	USer Interface eXtensible Markup Language
VB	Visual Basic
VBA	Visual Basic for Applications
VoiceXML	Voice Extensible Markup Language
WAP	Wireless Application Protocol
WML	Wireless Markup Language
XIML	eXtensible Interface Markup Language

Chapter 1 Introduction

1.1 Motivations

As a first step, we analyze the reasons for our interest in the design of dialogues; we examine the challenge of working on or modelling interactive dialogues, the complexity of dialogues for interactive applications and the sophistication of designing dialogues.

1.1.1 Challenge of Modelling Dialogues

Natural language is at the heart of human dialogue, probably the most frequently used communication channel ever. *Information Systems* (ISs) do not escape from this observation: probably the most important part of an IS today lies in its capabilities to communicate information quickly, precisely and in a reliable way. More particularly, the User Interface (UI) of this IS is also concerned as it is considered to be the primary way of communicating with end users, who do not necessarily speak the IS's language but their own language with their own dialogue.

Many aspects may influence the dialogue between a UI and its end users in any context of use [Cal03]: aspects related to the *end user* (e.g. native language, cultural background), aspects related to the *computing platform* (e.g. application type, operating systems, *Integrated Development Environment* (IDE) used, technical requirements) and aspects related to the environment in which the end user is carrying out her/his task with the IS (e.g. the location, the organization, the human factors of the corporate environment). Because of this diversity, designing any dialogue between a system and its end users remains a permanent challenge.

1.1.2 Complexity of dialogue

As known, computer applications progress constantly in term of complexity [Gat08, Han03]. In parallel, users' needs become vary increasingly in interactive application. Indisputably, human-computer interaction becomes sophisticated, multimodal and multi device. This reality can be justified by the fact that computers today reach unimaginable levels of performance in calculation and memory.

Computers used to work in milliseconds, then moved up to microseconds and now are approaching nanoseconds for logic operations and picoseconds for the switches and gates in chips. Currently, NASA scientists are working to solve the need for computer speed using light itself to accelerate calculations and increase data bandwidth. What they are accomplishing in the lab today will result in the development of super-fast, super-miniaturized, super-lightweight and lower-cost optical computing and optical communication devices and systems.

Human-Computer Interaction (HCI) is one of the fields most affected by the aforementioned evolution. Before continuing, let us recall that HCI is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them. Certainly, we note that the power of the computer never ceases to inspire HCI' researchers. For example, a team of researchers has lately developed a system that uses computer vision to replace standard computer mouse functions with hand gestures [Car98, Duc07]. The system is designed to enable noncontact HCI, so that surgeons will be able to make more effective use of computers during surgery.

1.1.3 Designing dialogue

Interactive applications implemented in context described above must be well-designed with the aim of facilitating application correction (i.e. to correcting errors/bugs in the application) or extension (i.e. adding new functionalities or modifying existing tasks). In short, developer (designer, analyst or programmer) needs to have powerful tools which can enable him to have a total control of its application. In others words:

- The developer must have a good requirements specification which defines 'the best vision': the target to aim at for throughout project. Skipping the specification phase, or not covering the details sufficiently, can lead to the same kind of misunderstanding between parties that can occur with an oral contract. Thus, having a good initial specification helps advance the subsequent design and implementation phases to successful conclusion. A good specification gives a well-defined target to aim for but it does not guarantee that the target will not move.
- Once the specification is written, the developer must design his/her projects by partitioning the system into individual parts; defining and documenting the interfaces between the individual parts; deciding on and documenting the architecture of his/her solution, and deciding on and documenting the toolbox, libraries or components to be used.

In the development process, user interface design is so essential because in the opinion of many developers, over half of the development time is spent on the user interface portion. Indeed, apart from the characteristics of the computers to be used, the choice is not always easy for good language/toolbox/libraries for a good user interface relative to an interactive application. Also, a good part of the code is related to user interface.

In addition, with the advent of the Internet, a series of unique challenges are posed for user interface design and development. New applications with innovative user interfaces are emerging (e.g. an e-commerce application with multiple user interfaces for personal digital assistants, Web, mobile telephone, etc.). For these applications, the developer does not necessarily know users' needs and stereotypes and/or cannot sit down with them and walk them through the system. Therefore, adaptation and customization are now parts of the software developer's job. It is already true that people find that about 80% of

software maintenance costs result from the problems users have with what the system does (usability), rather than from technical bugs.

User-centred design and usability testing are cost-effective solutions to this problem. So, easy to use (usability) oriented software development enhances human productivity and performance, reduces training time and costs, increases employee autonomy and performance, guarantees job quality due to uniform work practices as well as facilitates knowledge capitalization.

Before continuing, let us look in more deep at the notion of dialogue which constitutes our main theme.

1.2 What is dialogue or Behaviour

Now, let's understand what we mean by the term dialogue and fix the particularity of dialogues that we aim.

1.2.1 Generic definition

According to the Larousse dictionary, to communicate is to make common, to share or to transmit. Vivier [Viv96] states that the dialogue is a particular case of communication. Indeed, during a dialogue two or several entities interact together, often with the objective of producing an agreement. Thus, a dialogue supposes at least:

- A *transmitter*: the activated entity at a given moment of the dialogue. The entity who engages, who acts, at a certain moment of communication;
- A *receiver*: the non-activated entity at a given moment of the dialogue. The participating entities regularly exchange the roles of receiver and transmitter;
- A *message*: the unity of the emitted data or information;
- A *code*: the language and/or the jargon used as channel to pass the message;
- An *objective*: the goal of the message.

1.2.2 Particularity

Our research is focused particularly on dialogues whose entities for communication are respectively a human being and a machine. Thus, to avoid confusion with the concept of conversation, dialogue, that is more generic, we adopt the term behaviour in this thesis. Indeed, we are interested in behaviour, more precisely in the specification of actions and/or information exchange, of human and/or machine actors during the execution process of an interactive task (e.g. pushing on a remote control to change television channel, seeking information on the Internet with a web navigator, transcribing orally a text via a program of voice recognition, etc.).

It is easy to continue this list because of, firstly, the constant progress of communication and information technologies, and, secondly, the type of the application which required more and more data flow.

Moreover, the modern world is characterized by a remarkably rich evolution with regard to interaction technologies: on the one hand, traditional interaction technologies by graphical interface (windows, buttons, mouse, keyboards, sensors with wire or embarked, etc.), and, on the other hand, remote interaction, or sensitive interaction, which uses technologies containing sensors (distance, presence, displacement, sound, colour, temperature, etc.) of system of recognition per camera and computer, linked to systems of real-time analyses. Let us consider four examples that support our observations:

1.3 Illustrations

To fix ideas, let us consider four examples to illustrate the dialogue in interactive systems. These illustrations have the advantage of showing the diversity and the complexity of human-machine dialogues.

1.3.1 Disney Humanoid Robot

Disney aims to bring more physical interactions between visitors and its attractions machines. Disney Research Center have developed [Yam09, Yam10] a humanoid robot which has the capability of playing catch and juggling while still maintaining a safe distance between itself and participants - responding to entertainment robots in theme park environments which typically do not allow for physical interaction and contact with guests. An external camera system (ASUS Xtion PRO LIVE) is used to locate balls and a Kalman filter to predict ball destination and timing.

Figure 1. Disney's Humanoid Robot learns to Play

The robot's hand and joint-space are calibrated to the vision coordinate system using a least-squares technique, such that the hand can be positioned to the predicted location.

1.3.2 Ticket machine

The touch screens for the purchase of transport documents, train tickets for example, are characterized by a simple and easily communication in appearance. But, in-depth this machine offers a powerful functionality. Indeed, in the case of cash payment, the machine is able to recognize money, to compute (addition, subtraction or multiplication), to print the difference between the price of the requested transport document and the money received. In the case of bank card payment, the machine is able to start a banking order to request in real-time the debit of the client account. Any transaction error between the customer and the machine can have unfortunate consequences. For example, traveller may not have his/her ticket and thus miss his/her transport, or, the company could lose money because the machine debits insufficient funds.

Figure 2. Tokyo train ticket machine.

1.3.3 Surgery robot

In surgery, computers assist surgeons in the realization of a diagnosis or the most precise and least invasive therapeutic gestures possible.

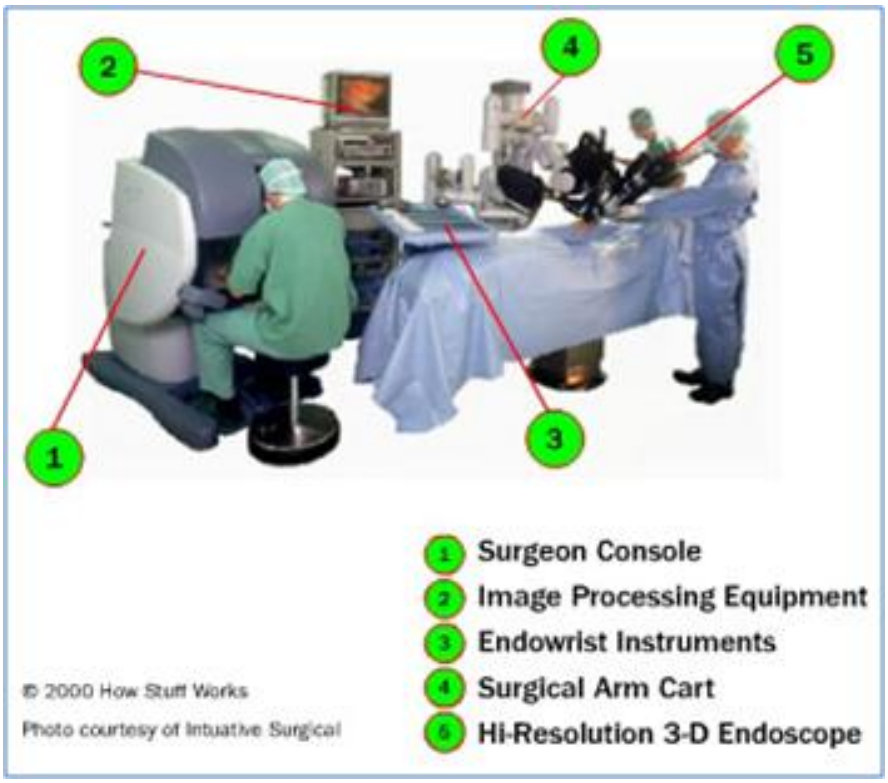


Figure 3. Surgery robot.

In such an environment, the interface introduced between the surgeon and the patient revolutionizes many aspects. As we can see in the images Figure 3, one of the robots used in surgery is composed of a console of surgery with stereo viewer with three-dimensional display incorporated, a carriage of surgery with arms of instrumentation and a carriage of imagery. The surgeon operates using two manipulators. On the screen of posting, the ends of the instruments are aligned with the manipulators to ensure the natural and foreseeable movements of the instruments.

1.3.4 Wii gameplay

The Wii gameplay revolutionizes human-machine interaction in video games. Its interface makes it possible to play golf by making real gestures, the swing for example. It is also possible to fight with genuine punches in the air. In terms of interface, there is a clear rupture compared to the other plays with the console or the screen only. Wii allows the combination of several widgets simultaneously. Indeed, the real revolution in this system is its controller, called the Wii Remote. Shaped like a TV remote, it has been designed to be used easily by beginners and pros alike. Sensors determine the Wii Remote's position in 3-D space, which means that racing-

game steering and a tennis swing, for example, are done through movements of the player's hand/arm rather than by just his thumbs.

Figure 4. Playing golf with Wii.

1.4 Dialogue aspects

Among all models involved in Model-Driven Engineering (MDE) of User Interfaces (UIs) of any interactive application in general, or for a web application in particular, the dialogue model is probably one of the most challenging remaining problems for several reasons that we can be organized in four categories described below.

1.4.1 Cognitive Aspects

- *Lack of ontological definition:* different terms, e.g. dialogue, navigation, behaviour, dynamics, conversation, the "feel", are inconsistently used to refer to the dynamic aspects of a UI, as opposed to the presentation, which refers to as the static aspects of a UI, e.g. its layout. We hereby define a *dialogue model* as the model that captures all dynamic aspects of user interface behaviour. This therefore includes dynamics at any level of any object that may appear in a user interface. This definition will lead us to define five particular levels later on.

- *Lack of actors*: a dialogue implies an exchange in real-time between two actors. That requires a good comprehension of each actor throughout the conversation. The great question is whether we can use this semantics of the dialogue when one of the actors

is a machine. In this context, the dialogue can be seen as a functionality by which a human operator can interact, handle, supervise or lead an automated system. The problem becomes complicated when the exchange relates to two machines. Within the framework of our research, we will see a dialogue like a network of nodes. Each exchange between actors must correspond to a passage from a node to another (possibly the same one). Then, dialogue scenarii would be the various possible courses in this network.

1.4.2 Conceptual Aspects

- *Lack of precise abstraction: in principle,* MDA suggests three levels of abstraction (i.e. computing independent model, platform-independent model and platform-specific model)[Rai04]. These three levels are rarely observed in the area of dialogue modelling where the platform-specific level remains predominant.

- *Lack of continuity: when two levels of abstractions are covered,* it is not always obvious to see how model-to-model mappings (whether achieved through transformations or not) are assured to establish and maintain continuity between them.

- *Lack of expressiveness:* the demand for more sophisticated dialogues calls for a dialogue model capable of accommodating the description of desired dynamic aspects, such as animations, transitions, the two traditional forms of adaptation (i.e. adaptability and adaptivity). A modern dialogue model should be expressive enough to model recent dynamic aspects.

- *Risk for modelling complexity:* it is likely that a more expressive model would tend to be more complex to define and therefore to use in general. The question would be to find the best abstraction: a modelling which can make it possible to graduate complexity, allowing the analyst to better understand and thus better control the problem.

1.4.3 Implementation Aspects

- *Lack of techniques combining genericity and flexibility*: developers lack techniques that would allow them to specify a user interface at an abstract, generic level, suitable for several platforms and contexts, while providing flexible, configurable adaptation to the specific target platforms.

- *Lack of performance:* how to reach information quickly when the size of the database exceeds hundreds of gigabytes. Database performance tuning has become a very important activity. The goal is to minimize the response time of queries and to make the best use of server resources by minimizing network traffic, disk Input/output and CPU time.

- *Lack of security:* it is essential for the developer to make safeguard his application; significant data must be protected. For the Web applications in particular, the encryption techniques are more than necessary.

1.4.4 Handling Aspects

- *Lack of advanced user interface support*: the ideal is that the representation to be made is as near as possible to reality. For example, playing tennis match with a Wii game, a ball cannot be represented by a bird. In the same way, for a weather chart an animation relating to a storm must be realistic. Thus, the user can very quickly interpret the danger without losing several minutes in the reading of the statistical data.
- *Lack of widget coverage*: the choice of the graphic components is very important. According to the type of an application and the context of its use, it is invaluable that the interactive be as real as possible. If not, the user cognitive effort will be too great which could entail errors that can have fatal consequences in critical applications.
- *Lack of user profile consideration:* It is known that for a given interface, a beginner user does not have the same behaviour as an expert. While working with a training application for example, user progression must be taken into account. Progressively as the user knowledge evolves, the system presents him with more advanced concepts.

1.5 Thesis

1.5.1 Thesis statement

Our objective is to build a methodology of designing and specifying User Interface (UI) behaviour. The aforementioned methodology must be at the same time structured, reproducible and independent of platform. It must also provide effective traceability for history management and its results will be reliable and demonstrable.

Firstly, we will remember that to specify a problem means to build methodically its statement as clearly as possible reducing to the maximum: ambiguities (words/terms with several meanings), contradictions (assertions being excluded one from the other), silences (absence or insufficiency of capital information), and the noises (amplification or exaggeration relative to not very useful information).

Methods of formal specification have the advantage of having a well-defined semantics. That makes it possible to work in a rigorous way and especially, valorisation (checking) supports end results compared to initial waiting [Pal94, Pat94]. It is partly true to believe that only the critical interfaces require a formal specification. As far as possible, it is always advised to specify any interface. Indeed, it rather often happens that developers spend much time when adding a simple button to an existing graphical window because the person at the origin of the interface is not present or if nobody knows which information is attached to which object. The situation becomes complicated when it is

necessary for an application to evolve to another programming language and/or platform. In these cases, the existence of a specification is not superfluous.

Secondly, we hereby refer to presentation as being the static part of a UI such as the description of all windows, dialogue boxes, widgets and their associated properties. In contrast, we hereby refer to behaviour as being the dynamic part of a UI such as the physical and temporal arrangement of widgets in their respective containers. The behaviour has also been referred to as dialogue, navigation, or feels (as opposed to look for presentation). Here are some typical examples of behaviours: when a language is selected in a list box, the rest of a dialogue box is updated accordingly; when a particular value has been entered in an edit field, other edit fields are deactivated because they are no longer needed; when a validation button is pressed, the currently opened window is closed and another one is opened to pursue the dialogue.

Indeed, for many years, the hardest part in conceptual modelling of User Interfaces has been its dynamic part. All other aspects, such as presentation, help, tutorial, etc. have received considerable attention and results, especially in model-based approaches and model-driven engineering.

The behaviour received limited attention for many reasons: declarative languages that have been typically used for modelling presentation are hard to use for modelling behaviour. Procedural languages could be used instead, but then induce a mixed-model-based approach that is complex to implement. Languages used for the final behaviour are very diverse (mark-up or imperative), hold different levels of refinement (ranging from simple properties to sophisticated behaviours), are hard to abstract into one single level of abstraction (especially for different platforms), are hard to implement for model transformation.

There is no consensus about what type of model should be used: some models exhibit a reasonable level of expressiveness but prevent the designer from specifying advanced behaviours, while other languages benefit from more expressiveness but are more complex to handle, especially for non-trained designers. Which appropriate modelling approach to take is also open: taking the greatest common denominator across languages (with the risk of limited expressiveness) or more (with the risk of non-support), especially because many different implementations exist based on code templates and skeletons, deterministic algorithms, graph transformation, etc.

Finally, we are unaware of any existing approach that consistently applies model-driven engineering principles for UI behaviour from the highest level (computing-independent model) to the lowest level (platform-specific model). Existing approaches only address some parts of some levels.

As the diagram below shows perfectly, the method we propose is based on a series of models. We will present each model in isolation before presenting the overall conceptual model of the methodology. Moreover, this model will be used to implement a software solution. The conceptual architecture and algorithms to exploit this software will be presented later.

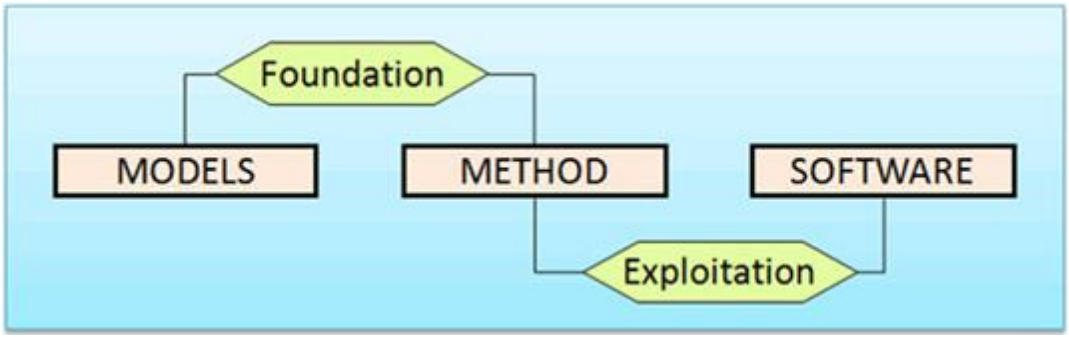


Figure 5. Methodological Approach.

We emphasize that this was no way to present a software solution for production. Our goal was to demonstrate the usefulness of different concepts and how to combine them to design and specify the behaviour of an interactive application.

Admittedly, there exist several solutions or attempts at solutions concerning behaviour specification. But their answer to the problem is often only partial. Within the framework of our research, we wish to propose a transform approach whose four elements constitute its characteristics:

1. It is based primarily on the concept of interface objects. The user has the choice between creating his/her own objects, used existing interactive objects and making both. However, it's important to determine which attributes, methods and events are necessary in dialogue script;
2. it gives a freedom concerning the level of specification. The user can choose to specify his interface at the abstract, concrete or final level;
3. it provides functionalities of passage intra and inter levels. The user could, for example, use the same abstract specification to provide two or several different concrete specifications. In the same way, it could start from a concrete specification to lead to another concrete specification by skews of the mappings;
4. it manages dialogue scripts traceability. It is possible to know who did what, which day and at what time. Also, if necessary, it is possible to cancel recent modifications, or simply to carry out an old version of a given script.

In order to address these objectives, we apply *Model Driven Engineering* (MDE) paradigm. The main characteristic is that each exploited model is a toolkit; a box of objects whose syntactic and semantic properties furnish dialogue scripts. Toolkits are classified

according to the levels of abstraction of the Cameleon Reference Framework: task and domain, abstract user interface, concrete user interface and final user interface. The dialogue modelled at the abstract user interface level can be reified to the concrete user interface level by model-to-model transformation that can in turn lead to code by model-to-code generation. Definite concepts are generals but in order to validate results, we limited ourselves to support three programming languages: Visual Basic, HTML Applications (HTA) and Microsoft Visual Basic for Applications (VBA). Two computing platforms are addressed: Microsoft Windows and Mac OS X. In this way, the approach demonstrates the capabilities of the abstractions in order to cover multiple programming paradigms and computing platforms. Five levels of behaviour granularity are exemplified throughout the methodology that is supported by a dialogue editor, a model transformer and a code generator integrated into one single authoring environment called Dialog Editor or Behaviour Editor.

The translation into UsiXML (USer Interface eXtensible Markup Language) dialogue scripts built in to this authoring environment produces an effective solution for describing user interfaces and their behaviour with various levels of details and abstractions without limit of device, platform, modality and context.

Therefore, we will defend the following thesis:

> *Apply Model-Driving Engineering paradigm to build an approach for designing multi-platform user interfaces dialogue.*

This methodology is model-based and is supported by a Dialog Editor, a model transformer and a code generator integrated into one single authoring environment. Also, regardless of the level specification, the developer has a single scripting language to manage the behaviour of an interface.

This way, scripts translation can constantly be done into UsiXML. As known, UsiXML describes user interfaces with various levels of detail and abstractions, depending on the context of use. UsiXML supports a family of user interfaces such as, but not limited to: device-independent, platform-independent, modality independent and ultimately context-independent. UsiXML allows the specifying of multiple models involved in user interface design such as: task, domain, presentation, dialogue and context of use, which is in turn broken down into user, platform and environment. Adding dialogue description from the above-mentioned authoring environment, UsiXML enriches a dialogue model.
The concepts introduced above are reviewed and defined in the next section.

1.5.2 Some Definitions

1.2.2.a Human-Machine design methodology

HCI becomes increasingly varied and complex. In this context, design methods in this field aim at putting together, in a harmonious way, theories and techniques, to help with the assisted design of a better user interface. As we will see, in the state of art there exist today several approaches in the design of the HMI. Fortunately, in one field or another, each approach offers additional advantages to the Designer. It is advisable to say that the most interesting design methods are those which are at the same time simple to use and completely in agreement with the experiment and the user's needs.

1.2.2.b Concrete User Interface

A Concrete User Interface (CUI) is defined as the abstraction of any Final User Interface (FUI) with respect to computing platforms, but with the interaction modality given. According to MDE, it is a platform-specific model (PSM). A CUI is made up of Concrete Interaction Objects (CIO), which are abstractions of widgets found in those platforms. Any CIO may be associated with any number of Behaviours. Behaviour is the description of an Event-Condition-Action (ECA) mechanism that results in a system state change. The specification of behaviour may be broken down into three types of elements: an event, a condition and an action. An event is a description of a run-time occurrence that triggers an action. The general format of an ECA rule is: (ON Event, IF Condition, THEN Action). The event specifies when the rule should be fired, the condition specifies the logical condition when it should be fired and the action precises what methods should be executed for this purpose. In other terms, we can say that a Concrete User Interface (CUI) abstracts an FUI into a UI definition that is independent of any computing platform. Although a CUI makes explicit the final look and feel of an FUI. CUI can also be considered as a reification of an AUI at the upper level and an abstraction of the FUI with respect to the platform.

1.2.2.c Abstract User Interface

An Abstract User Interface (AUI) is defined as the abstraction of any CUI with respect to interaction modality. According to MDE, it is a platform independent model (PIM). An AUI is made up of Abstract Interaction Objects (AIOs), which are abstractions of CIOs found in existing interaction modalities and linked through abstract relationships. Therefore, an AUI only specifies interaction between a user and a system in totally independent terms. Only later on, once the interaction modalities are selected and once the target computing platform is elicited, this AUI will be turned into CIOs and final widgets, respectively. Abstract Interaction Object (AIO) may be of two types Abstract Individual Components (AIC) and Abstract Containers (AC). An Abstract Individual Component (AIC) is an abstraction that allows the description of interaction objects in a way that is independent of the modality in which it will be rendered in the physical world. An AIC may be composed of multiple facets. Each facet describes a particular function an AIC may endorse in the physical world order to conciliate computer-support and human control. In others terms, an Abstract User Interface (AUI) abstracts a CUI into a

UI definition that is independent of any modality of interaction (e.g. graphical interaction, vocal interaction, speech synthesis and recognition, video-based interaction, virtual, augmented or mixed reality). An AUI can also be considered as a canonical expression of the rendering of the domain concepts and tasks in a way that is independent from any modality of interaction.

1.2.2.d UsiXML -User Interface eXtensible Markup Language

UsiXML (USer Interface eXtensible Markup Language), a User Interface Description Language aimed at describing user interfaces with various levels of details and abstractions, depending on the context of use. UsiXML supports a family of user interfaces such as, but not limited to: device-independent, platform-independent, modality independent and ultimately context-independent. UsiXML allows the specification of multiple models involved in user interface design such as: task, domain, presentation, dialogue and context of use, which is in turn broken down into user, platform and environment.

- UsiXML is precisely structured into four levels of abstraction that do not all need to be specified to obtain a UI.
- UsiXML can be used to specify a platform-independent, a context-independent and a modality-independent UI. For instance, a UI that is defined at the AUI level is assumed to be independent of any modality and platform. Therefore, it can be reified into different situations. Conversely, a UI that is defined at the CUI level can be abstracted into the AUI level so as to be transformed for another context of use.
- UsiXML allows the simultaneous specification of multiple facets for each AIO, independently of any modality.
- UsiXML encompasses a detailed model for specifying the dynamic aspects of UI based on productions (right-hand side, left-hand side and negative conditions) and graph transformations. These aspects are considered as the basic blocks of a dialogue model that is directly attached to the CIOs of interest.
- Thanks to these dynamic aspects, virtually any type of adaptation can be explicitly specified. In particular, a transformation model consisting of a series of adaptation rules can be specified equally in an integrated way with the rest of the UI.
- UsiXML contains a simplified abstraction for navigation based on windows transitions that is compatible with dynamics.
- UsiXML is based on Allen relationships for specifying constraints in time and space at the AUI level that can in turn be mapped onto more precise relationships at the CUI level. These relationships are applicable to graphical UIs, vocal UIs, multimodal UIs and virtual reality UIs.
- Similarly, a progressively more precise specification of the CIO layout can be introduced locally to concretize the Allen constraints imposed at the AUI level.

- UsiXML defines a wide range of CIOs in different modalities of use so as not to be limited only to graphical CIOs.
- UsiXML already introduced a catalogue of predefined, canonical inter-model mapping that can be expanded and taxonomy of task types that facilitate the identification and selection of concepts at both the AUI and CUI levels.

1.2.2.e Task Model

A task model describes the various tasks to be carried out by a user in interaction with an interactive system. After a comparison of several task modelling techniques, an extended version of ConcurTaskTree (CTT) has been chosen to represent the user's tasks and their logical and temporal ordering in the context of UsiXML. A task model is therefore composed of tasks and task relationships.

1.6 Reading Map

The remainder of this thesis is structured as follows:

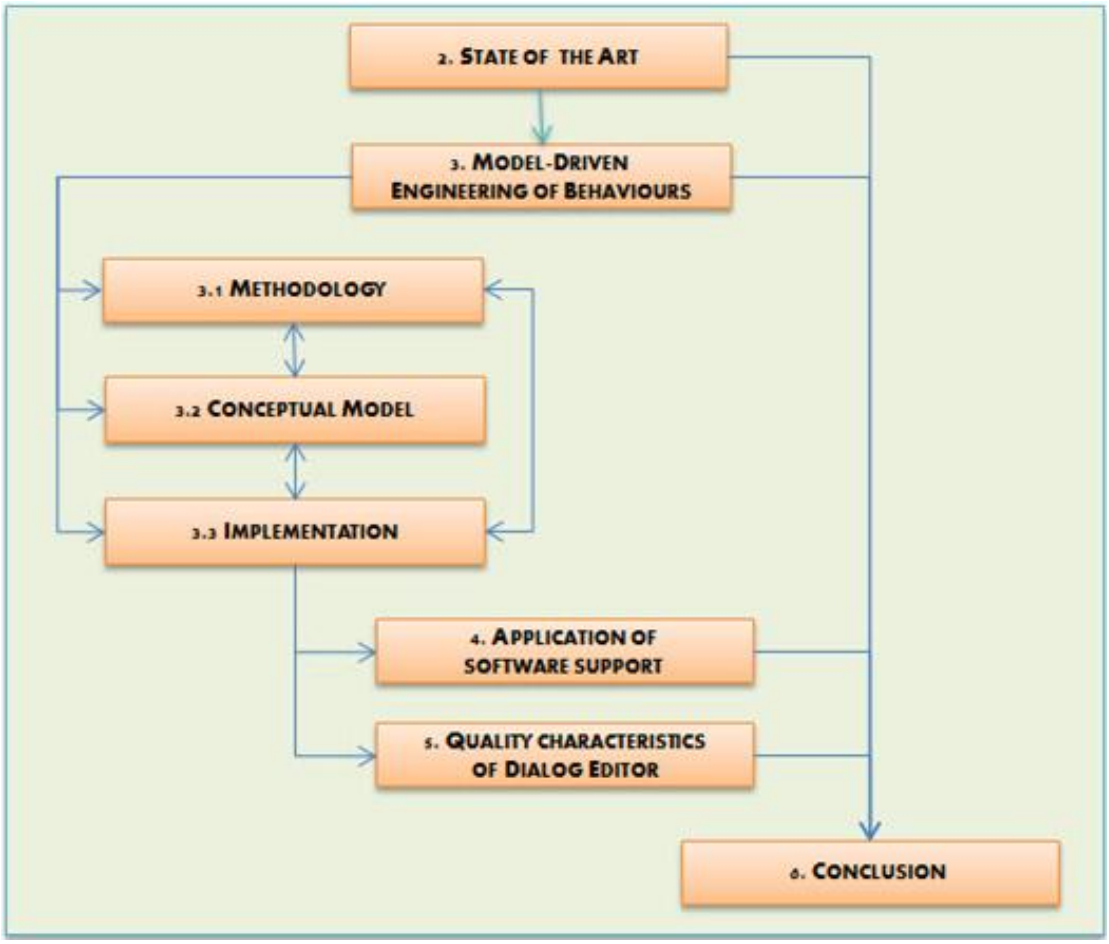


Figure 6. Reading Map.

Chapter 2, *State of the art*, recalls and presents a global view of methods, models, technical and tools which are used in dialogue specification. Particular emphasis is placed on abstract machines, user interface description language and UsiXML. The main Chapter 3, *Model-Driven Engineering of behaviour*, exploits the basic concepts of Chapter 2 to build our methodology. It will present the overall conceptual model and the generic algorithm to be applied to achieve the behaviour of a given interactive task. This chapter discusses method, models and software branches of methodology approach. Chapter 4, A*pplication of software support*, is the most practical of all. It presents two examples in which *Dialog Editor* was used; a simple and a more complex cases. Chapter 5, *Quality characteristics of Dialog Editor*, is based on ISO/IEC 9126 to examine technical, functional and interactive characteristics of the software that we implemented. Chapter 6, *Conclusion*, will summarize our contributions and explore some avenues for future work.

Chapter 2 State of the Art

The model concept is often used to abstract a technique, a method, an algorithm or simply a heuristics. In general, dimensions of models are reduced in order to facilitate their comprehension and their application.

Dialogue models enable reasoning about UI behaviour. Consequently, dialogue models are often considered as a continuation of task model concepts. This explains why the task model has been extensively used to derive a dialogue model, for instance, in an algorithmic way [Luy03] or in a logical way supported by model-to-model transformations [Sch07] and graph grammars [Goe96, Lim04]. We hereafter give a brief survey of dialogue modelling methods that percolated into the field of Human-Computer Interaction (HCI) development methods [Gre86, Lim04, Mba02, Van98, and Van03].

A very wide spectrum of conceptual modelling and computer science techniques has been used over the years to model a dialogue [Ari88, Bas99, Boo07, Bre09, Cac07, Car94, Cow95, Dit04, Elw96, Gre87, Har87, Jac86, W3C08, Mba00a, Mba00b, Mba99], some of them with some persistence over time, such as, but not limited to: Backus-Naur Form (BNF) grammars [Elw96, Jac86], state-transition diagrams in very different forms (e.g. dialogue charts [Ari88], dialogue flows [Boo07], abstract data views [Cow95], dialogue nets [Cle06], windows transitions [Van03]), state charts [Har87] and its refinement for web applications [Cac07], and-or graphs coming from Artificial Intelligence (e.g. function chaining graphs [Mba08]), event response languages and Petri nets [Bas99]. Some algorithms [Luy03] have also been dedicated to support the dialogue design through models, such as the Enabled Task Set [Pat09].

Rigorously comparing these models represents a contribution that is yet to appear. Green [Cle06] compared three dialogue models to conclude that some models share the same expressivity, but not the same complexity. Cachero et al. examine how to model the navigation of a web application [Cac07]. In [Cle06], the context model drives a dialogue model at different steps of the UI development life cycle.

So far, few attempts have been made to structure the conceptual modelling of dialogues in the same way as has been done for presentation, the notable exception being applying StateWebCharts [Win03] with Cascading style sheets [Win08] in order to factor out common parts of dialogues and to keep specific parts locally.

The DIAMODL runtime [Tra08] models the dataflow dialogue as Face Data Binding and includes extensions for binding EMF data to SWT widgets in order to link domain and dialogue models. Statechart logic is implemented by means of the Apache SCXML engine [W3W08], while GUI execution utilizes an XML format and renderer for SWT.

The Multimodal Interface Presentation and Interaction Model (MIPIM) [Sch05] could even model complex dialogues of a multimodal user interface together with an advanced control model, which can either be used for direct modelling by an interface designer or in conjunction with higher level models. Van den Bergh & Coninx [Van07] established a semantic mapping between a task model with temporal relationships expressed according to ConcurTaskTrees notation and UML state machines as a compact way to model the dialogue, resulting in a UML profile. Figure 14 graphically depicts some dialogue models in families of models.

Each family exhibits a certain degree of model expressiveness (i.e. the capability of the model to express advanced enough dialogues), but at the price of a certain model complexity (i.e. the easiness with which the dialogue could be modelled in terms specified by the meta-model).

We organize the rest of this chapter into three sections. The first deals with the modelling of dialogues by the use of abstract machines. The second explains dialogue management using UIDL (User Interface Description Languages) and the third gives some details on the characteristics of UsiXML.

2.1 Abstract Machines

Abstract machines, also known as mathematical models are used in the specification of dialogues since the pioneering work of Green [Gre86]. If we find it difficult to give an exhaustive list of these models, we intend to recall some definitions and basic concepts. This exercise will be useful because some of these models will be used in the methodology that we propose.

To illustrate the different abstract tools that we outline in this chapter, we use the example of a connection system that requires a login and a password. As in many systems, we assume that the user can make up to three attempts. This becomes interesting in the sense that each tool gives us the opportunity to emphasize one or more aspects of this problem of connection.

2.1.1 Backus-Naur Form (BNF) grammars

Backus-Naur Form, or BNF for short, is a notation used to describe context free grammars. The notation breaks down the grammar into a series of rules which are used to describe how the programming languages tokens form different logical units In computer science, BNF is a metasyntax used to express context-free grammars: that is, a formal way to describe formal languages. John Backus and Peter Naur developed a context free grammar to define the syntax of a programming language by using two sets of rules: i.e. lexical rules and syntactic rules.

BNF is widely used as a notation for the grammars of computer programming languages, instruction sets and communication protocols, as well as a notation for representing parts

of natural language grammars. Many textbooks for programming language theory and/or semantics document the programming language in BNF. There are many extensions and variants of BNF, including Extended and Augmented Backus–Naur Forms (EBNF and ABNF).

They are typically used to specify command languages [Gre86, Jac86]. Command languages express commands that modify the state of the UI at the user's initiative. Grammars are particularly good in detecting inconsistencies within command sets. An inconsistent UI may contain unordered or unpredictable interaction. Inconsistency renders the UI error prone and hard to learn. Reisner proposed an action grammar to describe Graphical UIs [Rei81]. Payne extended this grammar with their Task-Action Grammar (TAG) by covering three levels of inconsistency [Pay86]: lexical, syntactic and semantic. These established TAGs accuracy in predicting. Grammars are both efficient and effective for expressing sequential commands or users actions in general, but become complex for multimodality.

The actual reserved words and recognized symbol categories in the grammar represent "*terminals*". Usually, terminals are left without special formatting or are delimited by single or double quotes. Examples include: if, while, '=' and identifier.

In Backus-Naur Form, rules are represented with a "*nonterminal*" - which are structure names. Typically, nonterminals are delimited by angle-brackets, but this is not always the case. Examples include <statement> and <exp>. Both terminals and nonterminals are referred to generically as "symbols". Each nonterminal is defined using a series of one or more rules (also called productions). They have the following format:

- A rule consists of one or more productions;
- The production starts with a single nonterminal, which is the name of the rule being defined;
- This nonterminal is followed by a ::= symbol which means "as defined as". The ::= symbol is often used interchangeably with the symbol. They both have the same meaning.
- The symbol is followed by a series of terminals and non-terminals.

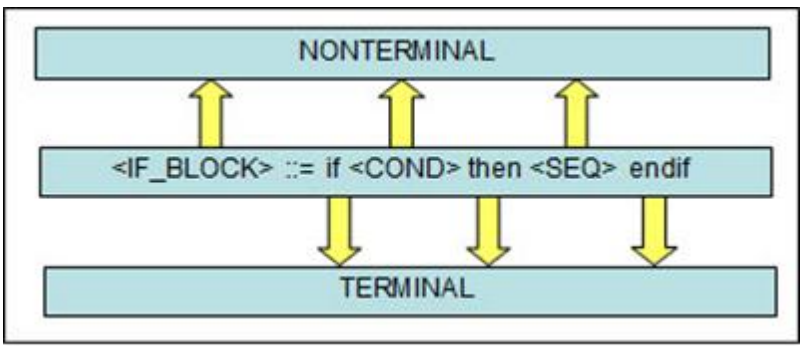


Figure 7. Sample of BNF Rule.

To return to the connection example, we can define a login as a string of lowercase alphabetic characters whose length does not exceed height. And, the password, a string that contains at least one lowercase letter, one uppercase letter, one number and one special character. Using regular expressions, we obtain:

```
<CONNECTION> ::= <LOGIN> <PASSWORD>
<LOGIN>    ::= ^[a-z]{8}?
<PASSWORD> ::= ^.*(?=.{4,10})(?=.*\d)(?=.*[a-zA-Z]).*$
```

Figure 8. Connection Sample; using BNF Rule.

2.1.2 State transition diagram

State transition diagrams are finite state machine representation that consists of a graph of nodes linked by edges [Gre86]. Each node represents a particular state of the system. Each edge species the input (i.e. event) required to go from one state to another. State transition diagrams have been subject to several extensions [Was85] and specializations, like Statecharts [Har87] that provide a means for specifying the dynamic behaviour of the interface. State transition diagrams present several drawbacks for modelling the UI. Indeed, today's UI tend to be modeless where one state can lead to many states. Furthermore this can be done using many different widgets of the UI. These two requirements match the quality criteria of reachability and device multiplicity.

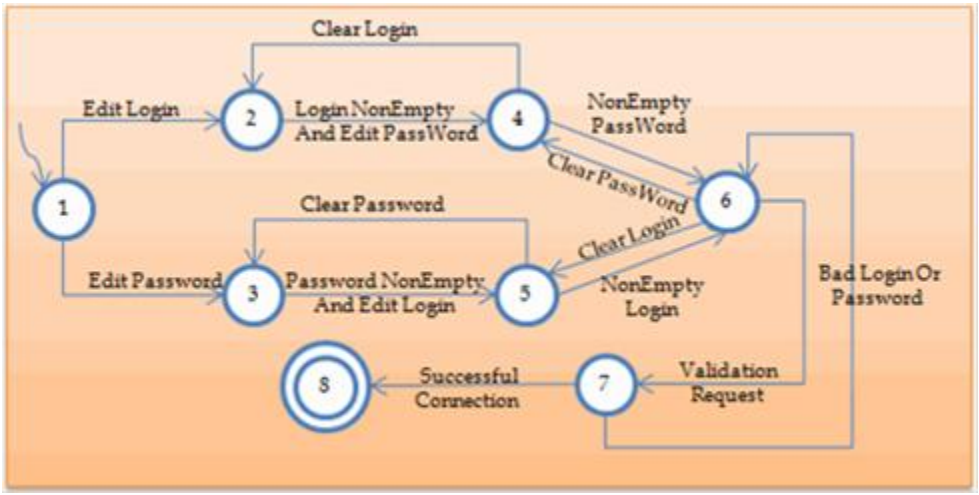


Figure 9. Connec tion Sample, State transition diagram.

In consequence, state transition diagrams are prone to a combinatorial explosion and tend to replace nodes by screen prints. In [Van03], the transition space is restricted to events and transitions that are triggered by window managers in graphical state transition diagrams, thus supporting only simple windows transitions [Mba02]. Many other forms of dedicated state transition diagrams have been extensively for dialogue modelling without knowing which one is superior to another: dialogue charts [Ari88], dialogue flows [Boo04,Boo05a,Boo05b], hierarchical dialogue flows [Boo08], interaction object

graph [Car94], Abstract Data Views [Cow95], dialogue nets [Jan93], models [Sch07,Sch05].

2.1.3 Statecharts

As for state transition diagrams, statecharts are supported by a graphical representation of dynamic aspects of systems [Har87]. There exists research which specifically addresses the modelling of UI behaviour with statecharts [Oli01,Pau99]. Statecharts represent state variables with rounded rectangles called states. State-changing mechanisms are represented with edges between states. State-changing is triggered by events and can be further conditioned. Statecharts facilitate the representation of state nesting, state history, concurrency and external interruptions. Statecharts [Har87] propose solutions to the shortcomings of state transition diagrams: statecharts have representational capacities for modularity and abstraction. The number of states with respect to the complexity of the modelled system increases more slowly with statecharts than with state transition diagrams. Statecharts avoid the problem of duplicating states and transitions. States in statecharts are hierarchical and capable of representing different levels of abstraction. Statecharts are more convenient for multimodal interfaces as they provide nesting facilities, external interrupt specification and concurrency representation. Statecharts have also been specialized for specifying the dialogue of web interfaces through StateWebCharts [Win03], that can be edited via a SWCEditor [Win05].

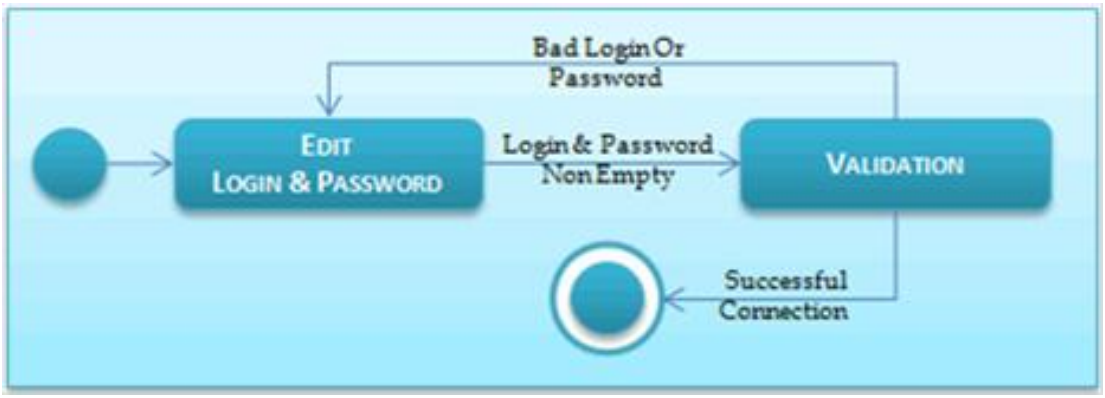


Figure 10. Connection Sample; Statechart diagram.

2.1.4 And-Or graphs

Borrowed from Artificial Intelligence, AND-OR graphs have been used to branch to various sub-dialogues depending on conditions, for instance in the EDGE system [Kle88]. And-or graphs have been expanded towards function chaining graphs [Bod95] by combining them with data flow diagrams [Van98].

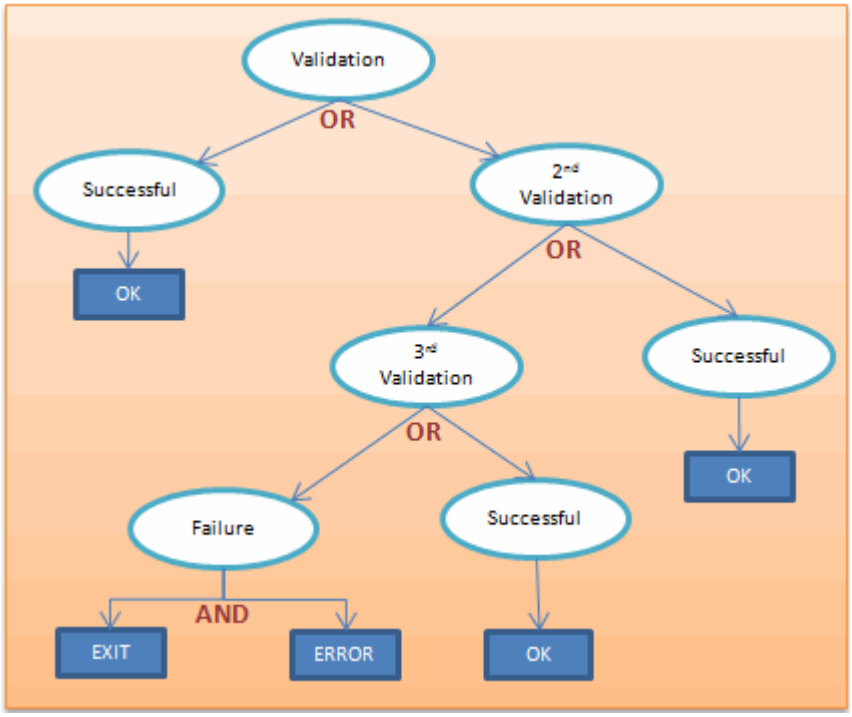


Figure 11. Sample Connection, And-OR Graph.

2.1.5 Event-Response Languages

Event-Response Languages treat input stream as a set of events [Hil86]. Events are addressed to event handlers. Each handler responds to a specific type of event when activated.

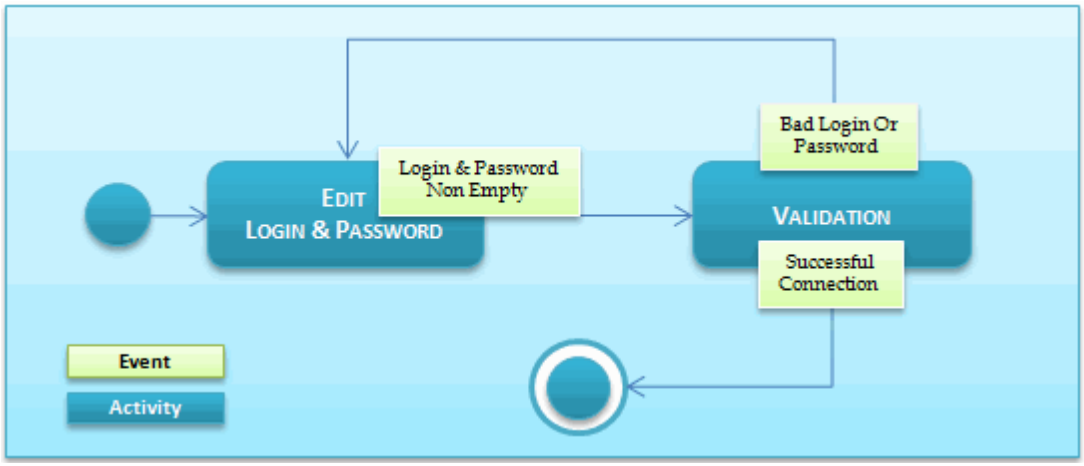


Figure 12. Sample Connection, Event-Response Diagram.

This type is specified in a condition clause. The body of the event generates another event, changes the internal state of the system or calls an application procedure. Several

formalisms are suited for event-response specification. They can be distinguished following their capacity to manage dialogue state variables and concurrency control. Production rules and pushdown automata [Ols84] are often used to describe event-response specifications.

2.1.6 Petri Nets

Petri nets are a graphical formalism associated with a formal notation. Petri nets are best suited to represent concurrency aspects in software systems. Petri nets represent systems with state variables called places (depicted as circles) and state-changing operators called transitions (depicted as rectangles). Connections between places and transitions are called arcs (represented by edges). States contain items called tokens (represented by black solid dots distributed among places). State change is the consequence of a mechanism called firing. A transition is red when all of its input places contain tokens. Firing involves the redistribution of tokens in the net, i.e. input tokens are withdrawn from input places and output tokens are added in output places. Like State Charts, Petri nets hold mechanisms to represent additional conditions and nested states. Petri nets have the advantage of being entirely formal. Thus, model checking of interest properties of the dialogue model could be applied [Pal94].

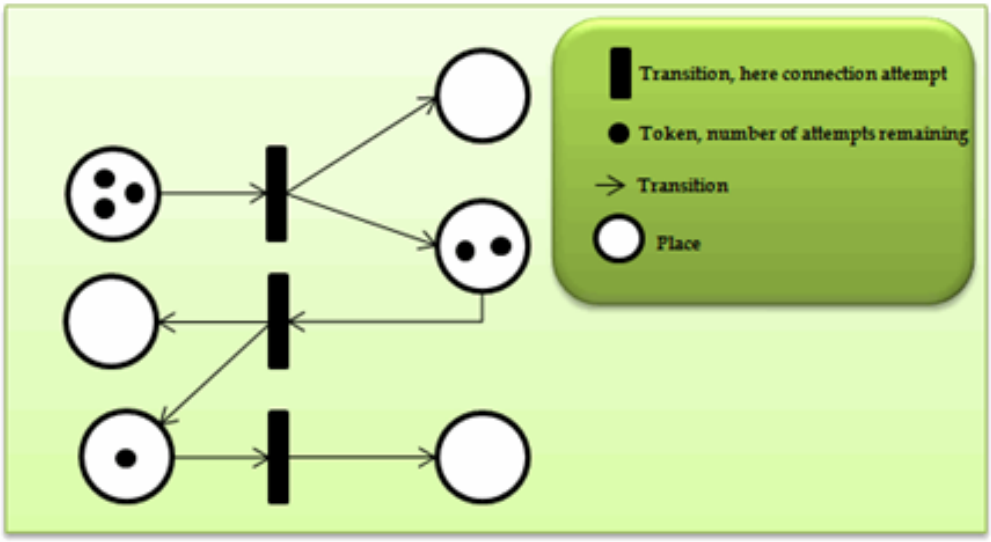


Figure 13. Sample Connection, Petri net.

Figure 14 graphically depicts most of these dialogue models in families of models. Each family exhibits a certain degree of model expressiveness (i.e. the capability of the model to express advanced enough dialogues), but at the price of a certain model complexity (i.e. the easiness with which the dialogue could be modelled in terms specified by the meta-model). At the left of Figure 14 relay BNF and EBNF grammars since they are probably the simplest dialogue models ever but they do not support many dialogue aspects. We can find respectively State Transitions Networks and their derivatives, then

Event-Response Systems. Petri nets are probably the most expressive models that can be used to model dialogues, but they are also the most complex to achieve. Therefore, we believe that we should not be as expressive and complex as Petri nets, but a little bit below. This is why we have selected Event-Condition-Action systems, one example being the DISL language [Sch05].

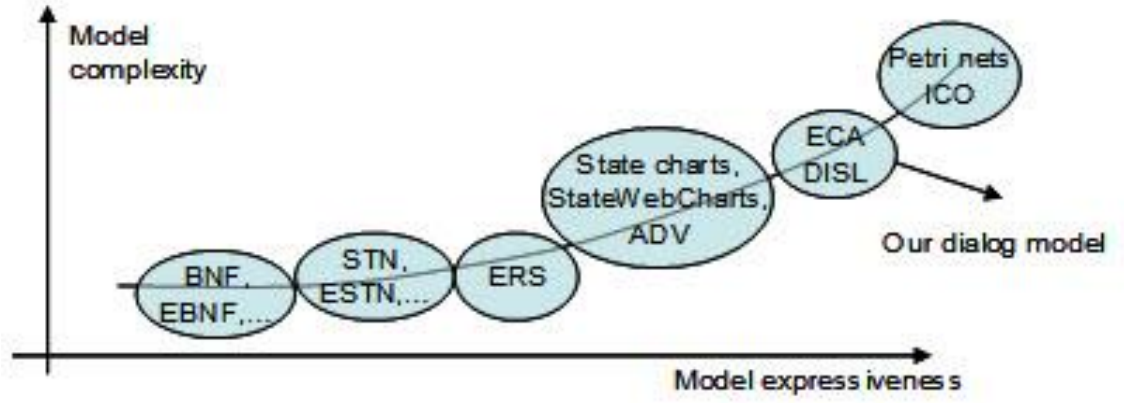


Figure 14: Model complexity as a function of their expressiveness.

2.2 Model-Driven Engineering

2.2.1 Introduction

The Model Driven Engineering, MDE for short, is a software engineering paradigm where models play a key role in all engineering activities (forward engineering, reverse engineering, software evolution…). In other words, MDE is a software design approach based on the concept of models and their transformation from one abstraction level to another or from one workspace to another [Bez04a, Bez04b]. The basic principle of MDE is "everything is a model", compared to the basic principle of object orientation "everything is an object".

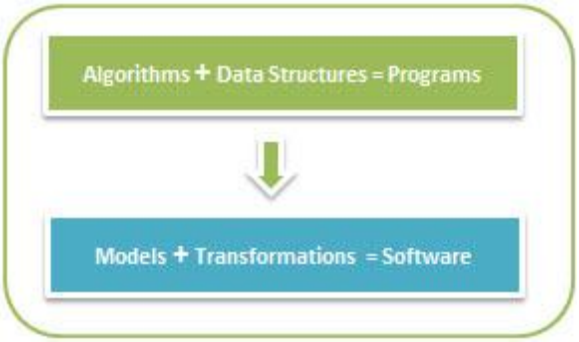


Figure 15. MDE Equation.

According to this definition, experts have revisited and transformed Niklaus Wirth's famous equation [Wir92] as shown in the Figure 15.

Before continuing this review, the main question is to fix what a "model" is. Indeed, in short, a model is an abstract representation of a system for some certain purpose and a meta-model is an abstract description of a model. The abstraction helps to neglect the less important aspects of a system, while concentrating on favourable parts that are desired to a specific study.

A model can come in many shapes, sizes, and styles. It is important to emphasize that a model is not the real world but merely a human construct to help us better understand real world systems. In general all models have an information input, an information processor, and an output of expected results.

A "model" can be seen as a measure, rule, pattern, example to be followed. In his book "*Allgemeine Modelltheorie*" (General Model Theory) [Sta73] Herbert Stachowiak describes the fundamental properties that make a Model:

- *Mapping*: Models are always models of something, i.e. mappings from, representations of natural or artificial originals that can be models themselves.
- *Reduction*: Models in general capture not all attributes of the original represented by them, but rather only those seeming relevant to their model creators and/ or model users.
- *Pragmatism*: Models are not uniquely assigned to their originals per se. They fulfil their replacement function a) for particular – cognitive and/ or acting, model using subjects, b) within particular time intervals and c) restricted to particular mental or actual operations.

Finally, among the numerous publications that we found on Model-Driven Engineering (MDE), we adopt the description given in [Bro11]. Indeed, the MDE approach to application design is an approach that makes use of several conceptual models so that each model manages one or more well-defined part(s) of the application. Because of the conceptual nature of such models, they would not have to address the technological problems associated with the final application handled by the users [Sch06, Bro09].

2.2.2 MDE objective

MDE is an open and integrative approach that embraces many other Technological Spaces in a uniform way. A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. Examples of technological spaces are Programming languages concrete and abstract syntax, Ontology engineering, XML-based languages, Data Base Management Systems (DBMS), Model-Driven Architecture (MDA), etc.

The goal of MDE is to increase both the short-term productivity, e.g. the amount of functionality delivered by a software artefact, and the long-term productivity, e.g. reducing the software artefacts' sensitivity for changes in personnel, requirements,

development platforms and deployment platforms [Bez05, Bro07, Omg05, Omg08, Sol00].

Therefore, MDE aims at defining models, methods and tools suitable for the precise and efficient representation of and reasoning concerning, software-intensive systems. MDE aims to encompass the entire life-cycle of a system, according to various dimensions such as system requirements, functionalities, data, processing, dependencies, architecture and infrastructure.

2.2.3 MDE Basic Principles

The idea promoted by MDE is to use models at different levels of abstraction for developing systems, thereby raising the level of abstraction in program specification. An increase of automation in program development is reached by using executable model transformations. Higher-level models are transformed into lower level models until the model can be made executable using either code generation or model interpretation.

Indeed, in the beginning of object technology, what was important was that an object could be an instance of a class and a class could inherit from another class. This may be seen as a minimal definition in support of object-oriented principle. We call the two corresponding basic relations *instanceOf* and *inherit*.

Very differently, what now seems to be important is that a particular view of a system can be captured by a model and that each model is written in the language of its meta-model. This may be seen as a minimal definition in support of MDA principle. We call the two basic relations *representedBy* and *conformantTo*.

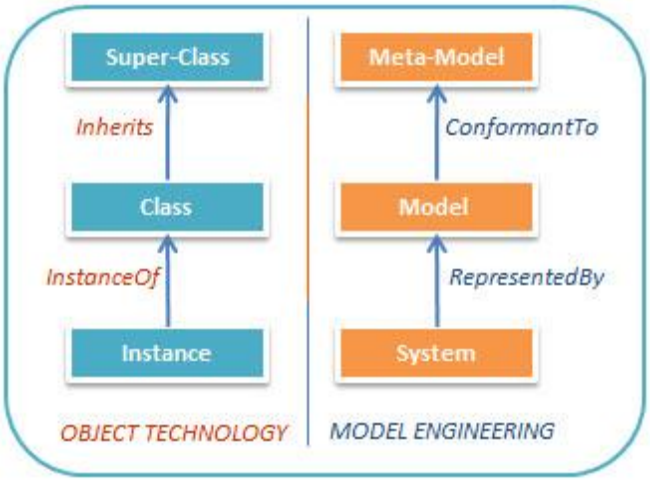


Figure 16. Object technology & Model engineering.

A model is specified in some model notation or model language. Since model languages are mostly tailored to a certain domain, such a language is often called a Domain-Specific

Language (DSL). A DSL can be visual or textual. A sound language description contains an abstract syntax, one or more concrete syntax descriptions, mappings between abstract and concrete syntax, and a description of the semantics. The abstract syntax of a language is often defined using a meta-model. The semantics can also be defined using a meta-model, but in most cases in practice the semantics aren't explicitly defined, they have to be derived from the runtime behaviour.

A model specified using a DSL is called a Domain-Specific Model (DSM). A complex system is usually described using multiple DSMs specified in different DSLs. These models refer to each other and have to be combined when executing them. Because complex systems ask for a lot of DSMs to model them, it is important to structure the modelling space.

As in each software engineering approach quality is an important aspect of MDE. Quality in MDE can be checked, or ensured, with three different techniques: model validation, model checking, and model-based testing.

In the construction of a dialogue specification methodology, we have chosen to operate MDE as one of the fundamental assumptions. We will show in the third Chapter how this hypothesis has been useful.

2.3 User Interface Description Languages (UIDLs)

A User Interface Description Language (UIDL) is a formal meta-language used in Human-Computer Interaction (HCI) in order to describe a particular User Interface (UI) independently of any implementation of this user interface.

Indeed, we indicated above, new classes of devices for accessing information have emerged along with an increased connectivity. In parallel to the proliferation of these devices, new interaction styles have been explored. Among these new styles are virtual reality, mixed reality, 3D interaction, tangible user interfaces, context-aware interfaces and recognition-based interfaces. As a result of this increasing diversity of devices and interaction styles, developers of next generation interfaces experience difficulties such as the lack of appropriate interaction abstractions, the need to create different design variations of a single user interface and the integration of novel hardware. As part of the user interface software research community effort to address these difficulties, the concept of UIDL, which has its foundations in user interface management systems and model-based authoring, has remerged as a promising approach. UIDLs allow user interface designers to specify a user interface, using high-level constructs, which abstract away implementation details [Abr99, Jac06, Mye00, Mor04, Sha07, The04].

Describing a UI via a UIDL does not assume that a particular implementation technology (e.g. programming language, markup language, dynamic programming, multi-paradigm programming) is required. As such, the UI does not assume the involvement of

only one interaction modality (e.g. graphical, vocal, tactile, haptic, multimodal) or interaction technique (e.g. drag and drop) or interaction style (e.g. direct manipulation, form fillings, virtual reality). A UIDL can be used during:

- Requirements analysis: in order to gather and elicit requirements pertaining to a UI of interest.
- Systems analysis: in order to express specifications that address the aforementioned requirements pertaining to a UI of interest.
- System design: in order to refine specifications depending on the context of use
- Run-time: in order to execute a UI via a rendering engine

A common fundamental assumption of most UIDLs is that UIs are modelled as algebraic or model-theoretic structures that include a collection of sets of interaction objects together with behaviours over those sets. This level of abstraction is commensurate with the view that the correctness of the UI presentation and behaviour takes precedence over all its other properties. UIDL specifications can be automatically or semi automatically converted into concrete user interfaces or user interface implementations.

A UIDL is more general than a User Interface Markup Language (UIML) that is often defined as [http://en.wikipedia.org/wiki/User_interface_markup_language]:
"A user interface markup language is a markup language that renders and describes graphical user interfaces and controls. Many of these markup languages are dialects of XML and are dependent upon a pre-existing scripting language engine, usually a JavaScript engine, for rendering of controls and extra scriptability."

As opposed to a UIML, a UIDL is not necessarily a markup language (albeit most UIDLs are) and does not necessarily describe a graphical user interface (albeit most UIDLs abstract only graphical user interfaces). A UIDL should necessarily be expressed as an XML dialect or bound to a particular scripting language.

There are today many UIDLs that could serve for modelling the behaviour of GUIs. Below is a list of some languages

2.3.1 Extensible Interface Markup Language (XIML)

XIML is an XML-based language that enables a framework for the definition and Interrelation of interaction data items. As such, XIML can provide a standard mechanism for applications and tools to interchange interaction data and to interoperate within integrated user-interface engineering processes, from design, to operation, to evaluation [Eise01, Puer02].

The XIML language is mainly composed of four types of components: models, elements, attributes and relations between the elements. We can distinguish two types of model, the interface model and the model components. The first is the root of any XIML document and contains the various sub-models (model components) available in XIML. The model components (task, domain, user, presentation, dialogue, platform, preferences and the general model) contain information specific to a dimension of the interface. Each model is composed of elements. Each model or element can possess features (composed of attributes or relations) or definitions (attribute or relation definitions).

2.3.2 Hypertext Markup Language (HTML)

HTML is the main markup language for displaying web pages and other information that can be displayed in a web browser. It is written in the form of HTML elements consisting of tags enclosed in angle brackets (like <html>), within the web page content.

HTML tags most commonly come in pairs like <h1> and </h1>, although some tags, known as empty elements, are unpaired, for example <img>. The first tag in a pair is the start tag, the second tag is the end tag (they are also called opening tags and closing tags). In between these tags web designers can add text, tags, comments and other types of text-based content. The purpose of a web browser is to read HTML documents and compose them into visible or audible web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page.

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts in languages such as JavaScript which affect the behavior of HTML webpages. Web browsers can also refer to Cascading Style Sheets (CSS) to define the appearance and layout of text and other material. The W3C, maintainer of both the HTML and the CSS standards, encourages the use of CSS over explicitly presentational HTML markup [Hako10].

There are five categories of elements for the HTML meta-model: elements specific to the head section (meta, script…), containers, (such as forms, tables…) that define hierarchy of elements, formatting tags (such as b, i, p…), lists (dl, ul…) and atomic tags that cannot contain other tags (img, object, button…) [Sten03].

2.3.3 Wireless Markup Language (WML)

Based on XML, WML is a markup language intended for devices that implement the Wireless Application Protocol (WAP) specification, such as mobile phones. It provides navigational support, data input, hyperlinks, text and image presentation, and forms, much like HTML (HyperText Markup Language). It preceded the use of other markup languages now used with WAP, such as HTML itself, and XHTML (which are gaining in popularity as processing power in mobile devices increases).

The root element of the model is a wml element, which can contain a meta, template or card node. The UI is contained in card elements and can be composed of navigation elements (+navelements), timer, paragraphs (p) or fields (+fields). Fields are broken down into several input elements (select, input…) and flow elements (+flow) that represent formatting tags for the text of the UI, such as b or i(for bold or italic text), tables, links(a) and images (img).

2.3.4 Voice Extensible Markup Language (VoiceXML)

VoiceXML's main goal is to bring the full power of Web development and content delivery to voice response applications, and to free the authors of such applications from low-level programming and resource management. It enables integration of voice services with data services using the familiar client-server paradigm. A voice service is viewed as a sequence of interaction dialogs between a user and an implementation platform. The dialogs are provided by document servers, which may be external to the implementation platform. Document servers maintain overall service logic, perform database and legacy system operations, and produce dialogs. A VoiceXML document specifies each interaction dialog to be conducted by a VoiceXML interpreter. User input affects dialog interpretation and is collected into requests submitted to a document server. The document server replies with another VoiceXML document to continue the user's session with other dialogs[McGl04].

The root of the meta-model of VoiceXML is a vxml element, which can contain meta information (meta and metadata), link, property, events handlers (+event handler), containers and input (+container) or executable content (+executable content). The VoiceXML UI is embedded in node belonging to the containers class. This superclass (preceded by a + symbol) groups logical containers (block, initial) and form-input elements (field, record, …) as they share common attributes and properties. The event handler superclass contains several predefined events (help, no input…) such as user-defined catchers (catch). Finally, executable content is the class for the rest of UI components. It contains tags allowing the modification of control flow (if, then, else, return…). such as output nodes (prompt, audio). Executable contents and event handlers are characterized by the fact that these elements cannot embed another element of the same class, contrary to containers.

2.4 UsiXML

The method that we propose to design and/or to specify user interfaces for multiple platforms is model-based. Therefore, it requires the use of a user interface description language (hereafter UIDL). This method is also transformational, as it consists of specifying a source UI, designed for the least constrained platform and then applying transformation rules to it to produce specific UIs targeted to more constrained platforms. These transformation rules will process different layers of the specification, according to the abstraction levels defined in the Unified Reference Framework described above. For this reason, the UIDL we will use needs to be structured in several layers. Until now, only a few UIDLs meet this requirement: XIML [Puer02], the last

versions of UIML [Ali03] and UsiXML. This section presents UsiXML and the conceptual content of this language. We focus on the UML diagrams used in UsiXML. Firstly, let us examine the data structure of the rich and complex language UsiXML, before developing each of its models.

2.4.1 Data Structure

The user interface description language UsiXML ([Limb04]) allows designers to describe various aspects of a user interface, while using the same language. Depending on the needs, a designer can adopt distinct viewpoints on the same user interface. In the early stages of design, he/she might choose to specify only high level functionalities (tasks) or domain objects. Later, the developer might want to give a very detailed description of the dialogue and presentation. These views on a user interface, called models in UsiXML, are organized in abstraction layers, following the Unified Reference Framework.

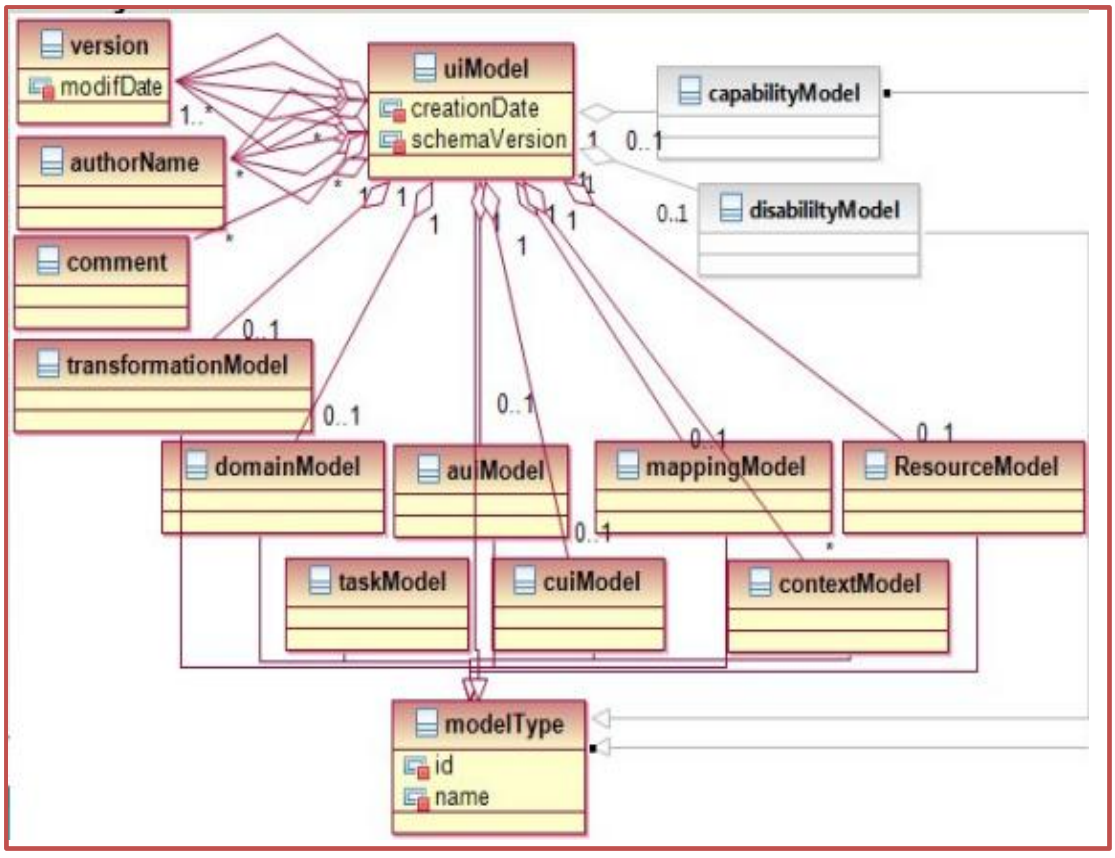


Figure 17. Constituent models in UsiXML.

A UsiXML specification is thus a combination of models. None of these models is mandatory and every combination of models is allowed. UsiXML is equipped with eight main types of model, as illustrated on Figure 17: a task model, a domain model, an AUI model, a CUI model, a mapping model, a context model, a resource model and a transformation model.

The *task* and *domain models* both belong to the Tasks and Concepts level of the Unified Reference Framework. The task model is a description of the tasks carried out by a user in interaction with the system, while the domain model is a description of the objects and classes viewed or manipulated by the user.

The *AUI model* (Abstract User Interface) lies at the next abstraction level in the Reference Framework. It is used to specify which group of tasks and domain concepts will be presented together (for example, in the same window or card).

The *CUI model* (Concrete User Interface) is a detailed specification of the appearance and behaviour of the UI's elements.

The *mapping model* serves to establish relationships between models or elements of models (for example, between a task belonging to the task model and the widget of the CUI that permits the execution of this task).

The *context model* consists of three sub models: a user model, an environment model and a platform model:

- The *user model* decomposes the user population into user stereotypes, described by attributes such as the experience with the system or with the task, the motivation, etc.
- The *environment model* describes any property of interest of the global environment where the interaction takes place. The properties may be physical (e.g. lighting or noise conditions) or psychological (e.g. level of stress).
- The *platform model* captures relevant attributes related to the combination of hardware and software where the user interface is intended to be deployed.

The *resource model* contains elements (title, tooltip, mnemonic...) specific to a given context (for example, the user's language). Resources are linked to objects of the CUI or AUI model.

Finally, the transformation model permits the specification of transformation rules under the form of graph transformation rules, taking advantage of the underlying graph structure of UsiXML. A graph transformation is expressed as a pair {LHS, RHS}, where LHS is the Left Hand Side of the rule and RHS is the Right Hand Side of a rule. LHS expresses a graph pattern that, if it matches the host graph, will be modified to result in another graph called resultant graph, according to what is specified in RHS [Limb04b]. This formalism supports different types of transformation: abstraction (e.g.; recovering an AUI model starting from a CUI model), reification (e.g., generating a CUI from a task model and a domain model) and translation (e.g., adapting a CUI designed for one specific context of use to another context of use). We will not rely on this formalism in this thesis, for two reasons:

- Some GD rules are inherently difficult to express using graph transformations. For example, it is far easier and more intuitive to express layout transformations by describing the algorithms used to generate the results than by giving a precise description of the pre- and post-conditions of the rule as patterns defined on a graph.

In particular, the difficulty in ordering the sub-steps of a given rule is a serious obstacle both for layout transformation rules and for the splitting rules.

- Even for simple transformations, such as modifying fonts size for example, relying on graph transformations has a negative impact on performance, because the process requires the collaboration of different tools, the use of several internal formalisms and several steps:

  1. Firstly, models are built using a graphical editor. These editors (IdealXML [Mont05], GrafiXML) possess an internal representation of the model and export it in UsiXML
  2. The UsiXML models are imported within AGG (Attributed Graph Grammar tool [Ehri99]), a graphical environment for specifying and executing graph transformations where the rules are applied to the graph structure
  3. The resulting models are exported from AGG to UsiXML.

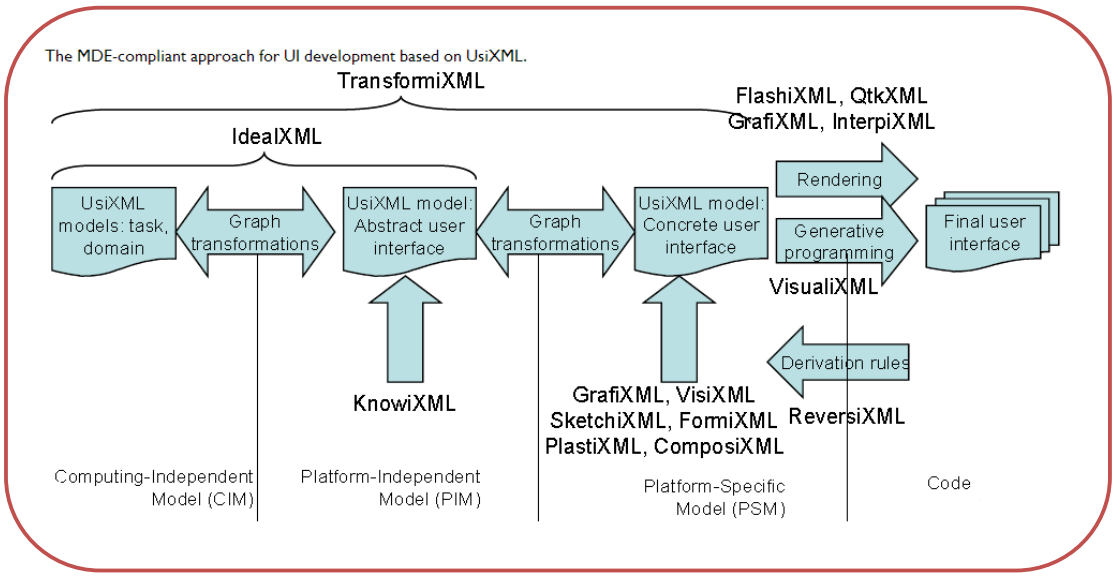


Figure 18. UsiXML Development Process

The next sections will be dedicated to a precise definition of the conceptual content of the models that are relevant in the framework of Graceful Degradation: task, domain, AUI, CUI, platform, interactor and mappings. The interactor model is a separate model that is not part of UsiXML. It permits the production of meta-descriptions of the toolkits available on a given platform. We will not make use of the other UsiXML models.

2.4.2 Task Model

A task model, as defined above, is a description of the tasks that a user will be able to accomplish in interaction with the system. This description is a hierarchical decomposition of a global task, with constraints expressed on and between the subtasks. The task model of UsiXML (see Figure 19) is an (slightly) extended version of ConcurTaskTree (CTT) [Pate00]: a hierarchical task structure, with temporal relationships specified between sibling tasks.

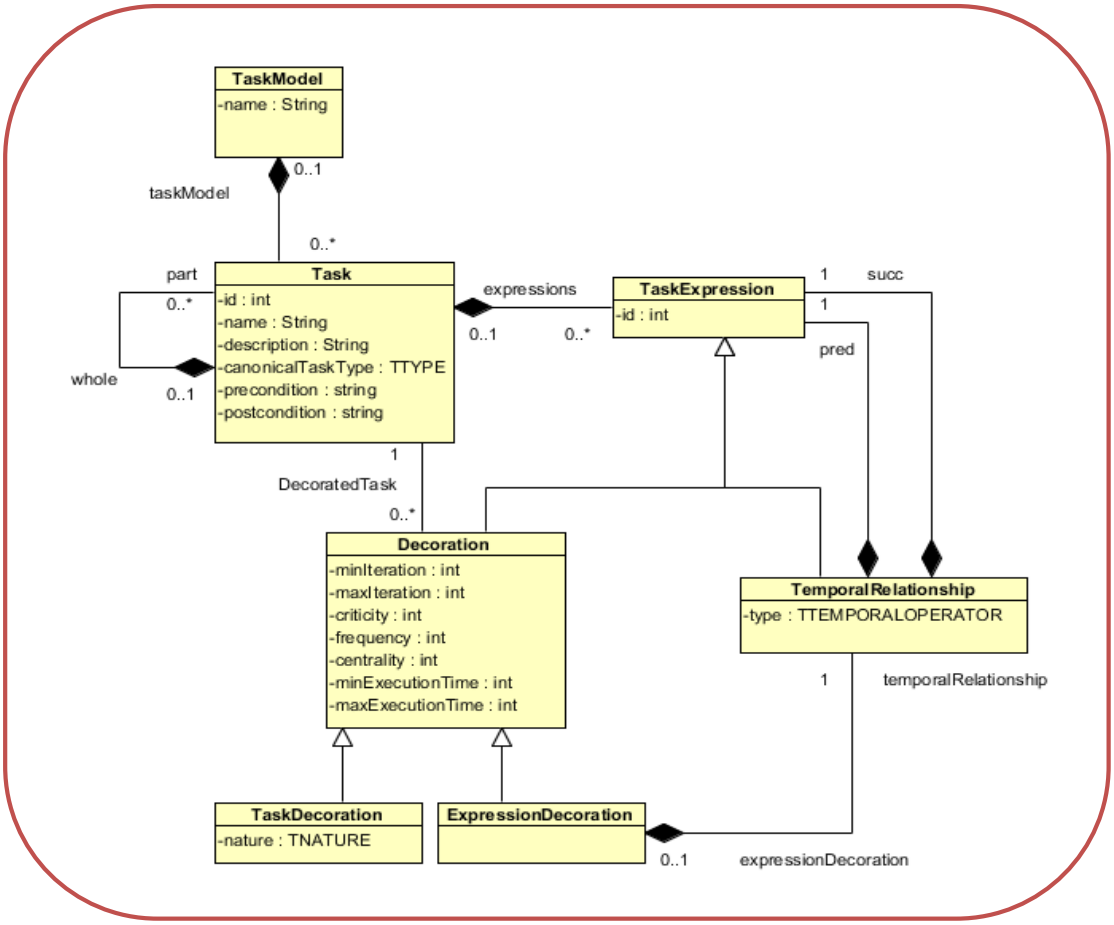


Figure 19. UsiXML Task Model.

2.4.3 AUI Model

An AUI model is an expression of the rendering of the domain concepts and tasks in a way that is independent from any modality of interaction. In UsiXML, the AUI (see meta-model on Figure 21) is populated by Abstract Interaction Objects and AIO Relationships.

Abstract Interaction Objects (AIO's) are elements populating the AUI. They may be of two types: *Abstract Containers* (ACs) and *Abstract Individual Components* (AICs).

Abstract Containers (ACs), also named *interaction spaces* or *presentation units,* define the grouping of tasks that have to be presented together, in the same window or page for

example. An abstract container contains other AIO's. It may be reified into graphical containers like windows or dialogue boxes.

Abstract Individual Components (AICs) are individual elements populating an abstract container. AICs are an abstraction of widgets found in most toolkits (for example windows, buttons or a vocal output widget in auditory interface).

An AIC may be composed of multiple *facets* describing the type of interactive tasks it is able to support. Each facet describes a particular function an AIO may assume. Four main facets have been identified:

1. An *input* facet describes the type of input that may be accepted by an AIO.
2. An *output* facet describes what data may be presented to the user by an AIO.
3. A *navigation* facet describes the possible container transition a particular AIO may enable.
4. A *control* facet describes possible methods of the domain model that may be triggered from an AIO.

Some AIO's may assume several facets at the same time (for instance, an AIO may display an output while accepting an input from a user).

AIO relationships are abstract relationships between two distinct AIO's. Our description of these relationships is more precise and complete than what can be found in the current UsiXML specification (introduction of new constraints, of new types of relationships). These proposals are intended to be included in the next UsiXML release.

AIO relationships indicate the existence of some spatio-temporal or logical setting among AIO's. A given pair of source and target AIO's can be linked by several AIO relationships. The operators between the abstract interaction objects in the TERESA tool [Pate02] or the abstract constraints expressed between components in some constraint-based automated layout systems [Lok01] are examples of the use of AIO relationships in the literature. Different types of AIO relationships can be defined:

- *Decomposition relationships* allow the specification of a hierarchical structure of abstract containers.
- *Spatio-temporal relationships* are modality-independent constraints between AIO's, using the temporal relationships defined by Allen [Alle83]. When UsiXML is used for specifying GUIs, they are redundant with the graphical relationships defined at the Concrete User Interface level: for this reason, we will not make use of Allen relationships in the context of this thesis.

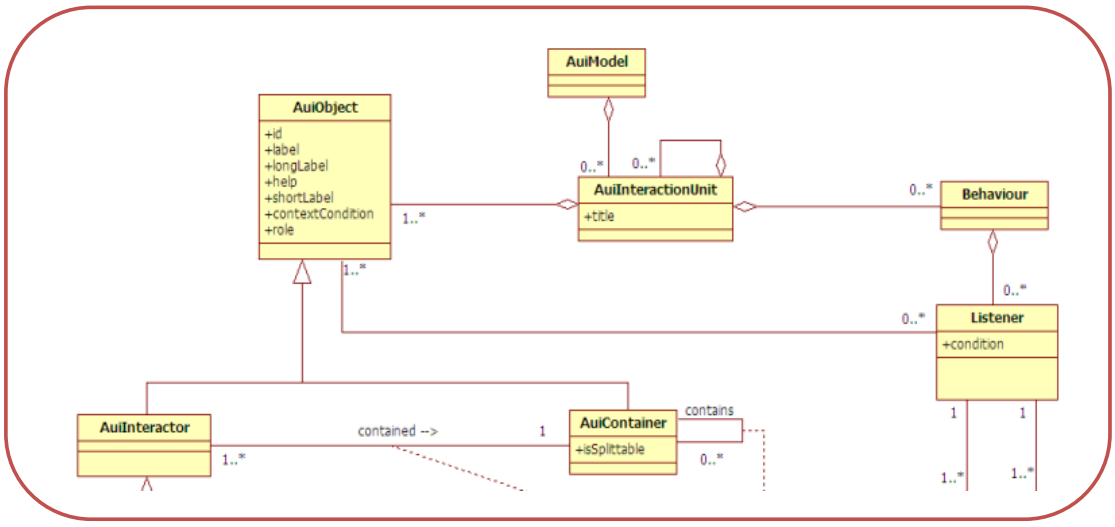


Figure 20. Top part of UsiXML Abstract User Interface.

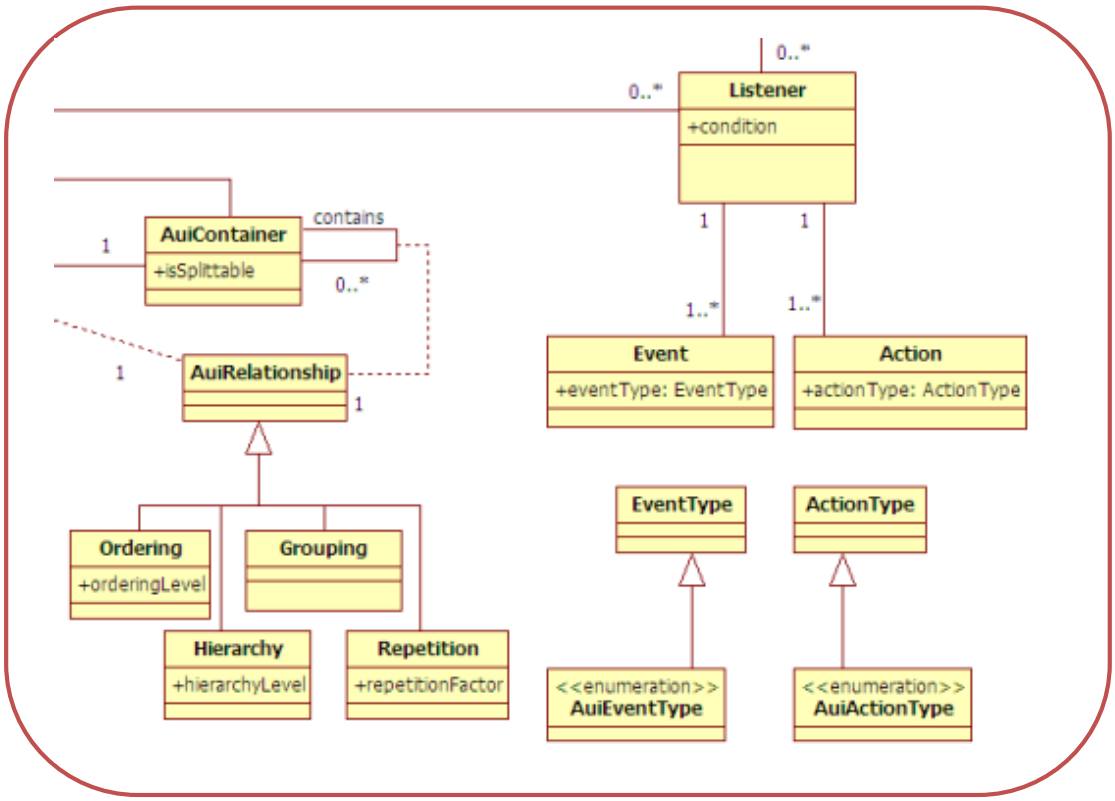


Figure 21. Lower right part of UsiXML Abstract User Interface

- *Abstract grouping* is an abstract relationship between two or more AIO's of the same abstract container that need to be grouped together, regardless of the actual layout that will be defined at the Concrete User Interface level.

- Conversely, *abstract separation* is an abstract relationship between two AIO's of the same abstract container that need to be separated from each other (for example, by a blank space or a separation line in graphical user interfaces, by a beep in auditory user interfaces...)
- *Differentiation* is an abstract relationship between two AIO's that should be differentiated from each other. For example, an "erase all" button could be differentiated from its neighbours, in order to avoid confusions.
- *Is-title-of* is an abstract relationship between one output AIO that represents a title and the AIO it describes.
- *Hierarchy* is an abstract relationship between two or more AIO's that form a hierarchy. For example, a series of titles in a document could be linked with a hierarchy relationship.
- *Abstract adjacency* is an abstract relationship between two AIO's that have to be adjacent (which is not possible to specify using Allen relationships).
- The *Order* relationship specifies some kind of ordering between two or more AIO's
- *Dialogue control* relationship allows a specification of a flow of control between the abstract interaction objects in terms of LOTOS operators.

2.4.4 CUI Model

A CUI Model represents a concretization of an AUI Model. A CUI is populated by *Concrete Interaction Objects* and *Concrete User Interface relationships* between them.

Concrete Interaction Objects (CIO's) are the building blocks of the CUI. They are an abstraction of widgets sets found in popular toolkits such as Java AWT/Swing or HTML4.0. UsiXML distinguishes between *graphical CIO's* and *auditory CIO's*. In the context of this thesis, we will only consider graphical CIO's. UsiXML further classifies graphical CIO's in two categories: *graphical containers* and *graphical individual components* (Figure 22).

A *graphical container* is a graphical CIO that can contain other CIO's, including other containers. UsiXML's metamodel contains a list of 11 types of containers: dialogue box, menu bar, menu pop-up, tabbed dialogue box and tabbed item, table and cell, tool bar, status bar, window and box.

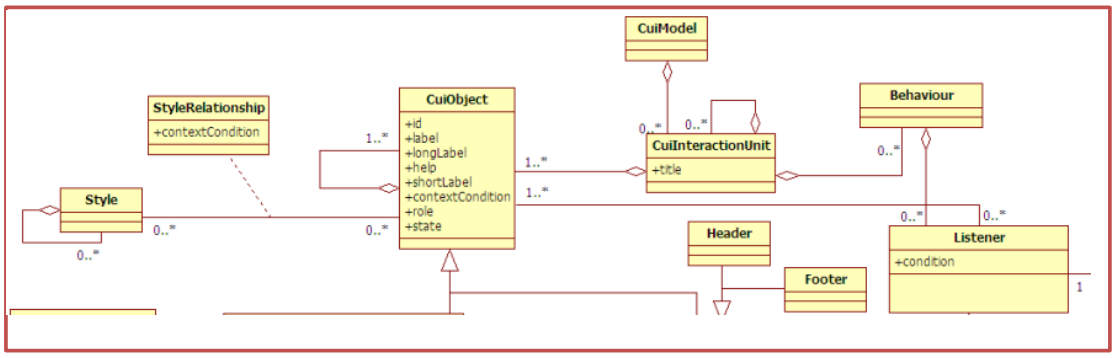


Figure 22. Top part of CUI Model in UsiXML.

A *graphical individual component* is a CIO that permits the observation or the manipulation of domain objects, or the calling of domain methods. Graphical individual components are a direct abstraction of widgets found in popular toolkits. For example, UsiXML's *checkBox* component corresponds to <INPUT TYPE = CHECKBOX> in HTML 4, JCheckBox in Java Swing or Checkbutton in Tcl/Tk. The list of graphical individual components in UsiXML includes *text component*, *video component, image component*, *button*, *toggle button*, *radio button*, *checkbox*, *combobox*, *listbox*, *spin*, *menu items drawing canvas*, *colour picker*, *date picker*, *file picker*, *hour picker*, *progression bar* and *slider*.

Concrete Interaction objects are linked by Concrete User Interface relationships. Again, they are divided into *auditory relationships* and *graphical relationships*. *Dialogue control relationship* can be defined between both types of interaction objects.

Graphical relationships express different types of constraints between a source graphical CIO and a target graphical CIO:

- *Relative positioning constraints* specify a positioning relationship between two components. Most of these constraints are a concretization of Allen relationships for graphical UI's: insertion, left-of, right-of, superiority, inferiority. Other constraints were impossible to express at the AUI level: left-indentation, right-indentation, horizontal adjacency and vertical adjacency.
- *Graphical transitions* specify a transition between two containers. Transition types are open, close, minimize, maximize, suspend/resume.
- *Alignment relationships* specify a relationship between two components and a guide extending their edges (vertical alignment, horizontal alignment) or crossing their centre either horizontally (horizontal centred alignment) or vertically (vertical centred alignment). With the exception of centred alignment, these relationships have direct correspondences at the AUI level (i.e.; they can be expressed in terms of Allen relationships).
- *Adjacency relationships* indicate that there is no interpolated component between two graphical CIO's, either in the horizontal direction (horizontal adjacency) or in the vertical direction (vertical adjacency).

Dialogue control relationships allow a specification of a flow of control between the concrete interaction objects, independently from the task model, using LOTOS operators. Dialogue control relationships at the CUI level are a refinement of the dialogue control relationships defined at the AUI level.

Relative positioning constraints (e.g.; left-of, inferior-to...) between two components can also be specified by the type of *box* that contains the CIO's. *Boxes* are the basic layout mechanism in UsiXML. A box can contain other boxes or graphical individual components. Boxes are characterized by:

- Their type: horizontal, vertical, grid.
- Their relative width and height with respect to their parent container.
- Information on their resizability and their minimum width and height.
- Optional balance constraints.
- A "splittable" attribute that indicates whether the box may be redistributed between several abstract containers.

UsiXML's Concrete User Interface is a hybrid model that contains at the same time information on the presentation of the UI and on its behaviour. At the CUI, each CIO can be linked to a *behaviour*. A *behaviour* is the set of reactions of the user interfaces to *events* such as user interactions, changes in the system state, period of time elapsed... These events trigger *actions*, such as a method call or a transition to a target container, provided that certain *conditions* are met.

2.4.5 Dialogue Model

UsiXML's dialogue model is the ultimate goal of research conducted in the context of this PhD investigation.

Indeed, the main chapter 3 describes a model-driven engineering approach for specifying, designing, and generating consistent behaviours in graphical user interfaces in multiple contexts of use, i.e. different users using different computing platforms in different physical environments. This methodological approach is structured according to the levels of abstraction of the Cameleon Reference Framework: task and domain, abstract user interface, concrete user interface, and final user interface. A behaviour model captures the abstractions of the behaviour in terms of abstract events and abstract behaviour primitives in the same way a traditional presentation model may capture the abstraction of the visual components of a user interface. The behaviour modelled at the abstract level is reified into a concrete user interface by model-to-model transformation. The concrete user interface leads to the final user interface running thanks to code by model-to-code generation.

2.5 Conclusion

2.5.1 Overview

It is true that the list we offer on the state of the art is not exhaustive but this list has the advantage of reflecting the evolution of our research.

Indeed, firsly, the initial objective was to propose a methodology that is demonstrable, generic and reproductive. Under these conditions, the mathematical models described above are best placed to help us to achieve these goals. The results related to Windows Graphical Notation[Mba02] confirm these initial choices.

Secondly, insofar as we want a methodology that can help move from one level of abstraction to another without losing information, it was obvious that we exploit the model-driven engineering to express built concepts and models. We summarize the benefits of operating MDE in the following table:

Thirdly, to better manage scripts dialogues, especially the passage of a script from one level of abstraction to another level, we need to construct a description language. This reflects our interest in the interface description languages in order to better understand the functioning and the peculiarity of each of them. We are limited to considering only four examples to cover a broad scope.

Finally, our research is conducted within the context of the project UsiMXL. the ultimate goal is to integrate the results of the methodology to build in UsiXML environment. Thus, our work has been carried out so as to better understand UsiXML.

In the following table, we recall some interesting properties concerning the dialogues. Then, we will discuss how some formalisms support these properties.

Table 1. Dialogues Properties

PROPERTY	DESCRIPTION
COMPLEXITY	Human, machine or data resources needed for interacting with the computer to accomplish the task.
REUSABILITY	The ability to reuse relies in an essential way or the ability to build larger things from smaller parts, and being able to identify commonalities among those parts
COMPLETENESS	Ability to take into account all the possibilities of interaction. For example, in the context of abstract machine and; looking at each state, is there an arc coming from each state for each possible user action? If not, what is the effect on the system if the user performs this action?

DETERMINISM	Is the behaviour uniquely defined for each user action? In a simple abstract machine this corresponds to checking that there is at most one arc labelled with each user action from a particular state.
CONSISTENCY	Does the same user action have a similar effect in different states? If not are these dialogue modes visibly different?
REACHABILITY	Can anywhere from be reached from anywhere else? For example, you are at a particular dialogue state and you want to get to a different state to reach. Is there a sequence of user actions which is guaranteed to get you there? In addition, we may want to ask just how complicated and long that sequence is.
REVERSIBILITY	The ability to cancel an interactive action. Imagine you have just carried out an action, but wish you had not. This is a special case of reachability, but one which we expect to be especially easy — we all make mistakes. Note this is not Undo — returning to a previous dialogue state does not in general reverse the semantic effect.
ADAPTABILITY	Ability of a dialogue to adapt itself efficiently and fast to changed circumstances. The objective is to determine if there are technical and / or functions to transform a dialogue from one environment to another or from one abstraction level to another without losing information and interaction.
SCABILITY	Ability of a system, network or software to adapt to handle an increasing volume of work or data.

Now, without being exhaustive, let us enumerate some formalisms used in the specification and/or the design of dialogues with the objective of fixing the level with which they handle the properties listed above. To achieve this goal, we use the five indicators:

Very Good
Good
Medium
Bad
Very Bad

Table 2: Dialogue Formalisms Vs. Dialogue properties

FORMALISM FOR BEHAVIOUR		BRIEF DESCRIPTION	DIALOGUES PROPERTIES								
			COMPLEXITY	REUSABILITY	CONSISTENCY	SCALABILITY	COMPLETENESS	DETERMINISM	REACHABILITY	REVERSIBILITY	ADAPTABILITY
ABSTRACT MACHINES	Backus-Naur Form (BNF) grammars	BNF is a family meta-syntax notations used for expressing context-free grammars. The BNF uses the symbols (<, >, |, ::=) for itself, but does not include quotes around terminal strings. This prevents these characters from being used in the languages, and requires a special symbol for the empty string	Bad	Good	Good	Good	Bad	Very Good	Medium	Bad	Bad
	State transition diagram	State transition diagrams are used in modelling systems which can be described as a collection of discrete states. The machine receives events from the outside world, and each event can cause transition from one state to another. Traditional state machine modelling is based on sequential transitions from one state to the next. With this limitation, concurrent systems cannot be modelled.	Bad	Bad	Good	Good	Bad	Very Good	Very Good	Very Good	Good
	Statecharts	Statecharts are a graphical language to describe the behaviour of a discrete-state system. They are based on the exchange of messages, or events, between the system and its environment. Statecharts can model hierarchy and concurrency systems. The difficulty of using statechart is proportional to the number of states.	Medium	Bad	Good	Good	Medium	Very Good	Very Good	Very Good	Bad
	Event-Response Diagram	Event-Response Diagrams are used to specify sensors and interactions. These diagrams are used to show the relationships between events and tasks and how the events affect each other.	Bad	Bad	Medium	Medium	Bad	Good	Medium	Bad	Bad
	Petri Nets	A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, signified by bars) and places (i.e. conditions, signified by circles). Petri nets techniques can be used to examine the behaviour of the process and to calculate its performance measures.	Good	Bad	Good	Good	Good	Very Good	Very Good	Very Good	Bad

FORMALISM FOR BEHAVIOUR		BRIEF DESCRIPTION	DIALOGUES PROPERTIES								
			COMPLEXITY	REUSABILITY	CONSISTENCY	SCALABILITY	COMPLETENESS	DETERMINISM	REACHABILITY	REVERSIBILITY	ADAPTABILITY
FORMAL METHODS	Model based(Z,VDM)	Systems are modelled using sets and relations between sets or set theory. Vienna Development Method (VDM) and Z notation are the most widely used notations for developing model-based specifications [JON80, JON86, HAY86, SPI92].	Medium	Very Good	Good	Good	Good	Good	Medium	Very Good	Bad
	Algebraic (OBJ, Larch, ACT-ONE)	An algebraic specification does not try to build up a picture of the components of an object, but merely describes what the object is like from the outside. For an interface specification this sounds like a good thing, as we want to talk about the behaviour of a system from the user's viewpoint, not the way it is built. There are a wide number of algebraic specification notations including OBJ, Larch and ACT-ONE [DIX98].	Medium	Very Good	Good	Good	Medium	Good	Medium	Very Good	Bad
MODEL-DRIVEN APPROACH	Model-Driven Engeneering	Model Driven Engineering, aims to use models at different levels of abstraction for developing systems, thereby raising the level of abstraction in program specification. An increase of automation in program development is reached by using executable model transformations. Higher-level models are transformed into lower level models until the model can be made executable using either code generation or model interpretation	Good	Good	Bad	Good	Good	Medium	Bad	Bad	Very Good
	Model-Driven Architecture	Model-Driven Architecture approach defines system functionality using a platform-independent model (PIM) using an appropriate domain-specific language (DSL). The PIM is translated to one or more platform-specific models (PSMs) that computer can run. This requires mappings and transformations and should be modelled too.[OMG00]	Good	Good	Bad	Good	Good	Medium	Bad	Bad	Very Good

FORMALISM FOR BEHAVIOUR		BRIEF DESCRIPTION	DIALOGUES PROPERTIES								
			COMPLEXITY	REUSABILITY	CONSISTENCY	SCALABILITY	COMPLETENESS	DETERMINISM	REACHABILITY	REVERSIBILITY	ADAPTABILITY
UI DESCRIPTION LANGUAGE	XIML (eXtensible Interface Markup Language)	XIML is a simple markup language enabling functionality previously available only with complex programmed applications. It is a universal technology well suited for building full featured websites, mini-sites, widgets, web/mobile/PDA applications, GUI in desktop applications, touchscreens, etc.	Medium	Good	Medium	Good	Bad	Bad	Bad	Bad	Bad
	HTML(Hyper Text Mark-Up Language)	HTML is what is known as a "mark-up language" whose role is to prepare written documents using formatting tags. The tags indicate how the document is presented and how it links to other documents.	Medium	Good	Medium	Good	Bad	Bad	Bad	Bad	Bad
	WML (Wireless Markup Language)	WML is a markup language intended for devices that implement the Wireless Application Protocol (WAP) specification, such as mobile phones. It provides navigational support, data input, hyperlinks, text and image presentation, and forms, much like HTML	Medium	Good	Medium	Good	Bad	Bad	Bad	Bad	Bad
	Voice XML (Voice Extensible Markup Language)	VoiceXML is an application of the Extensible Markup Language (XML) which, when combined with voice recognition technology, enables interactive access to the Web through the telephone or a voice-driven browser. An individual session works through a combination of voice recognition and keypad entry.	Medium	Good	Medium	Good	Bad	Bad	Bad	Bad	Bad
	UsiXML (User Interface eXtensible Markup Language)	UsiXML is a formal Domain-Specific Language (DSL) used in Human-Computer Interaction and Software Engineering in order to describe any user interface of any interactive application independently of any implementation technology. A user interface may involve variations depending on: the context of use (in which the user is carrying out her interactive task), the device or the computing platform (on which the user is working), the language (used by the user), the organization (to which the user belongs), the user profile, the interaction modalities (e.g., graphical, vocal, tactile, haptics).	Medium	Good	Medium	Good	Bad	Bad	Medium	Bad	Very Good

2.5.2 Concerns

Information gathered above on the state of the art regarding interactive dialogues highlight the following shortcomings:

Concern#1: **Lack of methodology**

Behaviour is often left out for the profit of the presentation. Indeed, if there is a lot of work on the presentation, it is sorely lacking techniques or methods for construires interactive dialogues.

Concern#2: Lack of managing complexity

The unceasing advance of computer media is proportional to the increasing complexity of interactive applications. It is important that research be conducted to provide techniques for managing this complexity.

Concern#3: Lack of modelling

Behaviour is often programmed, not frequently modeled nor represented. When behaviour is represented, many different techniques exist.

Concern#4: Lack of computing-independent

Behaviour is hard to abstract from computing platform and from interaction modality. Indeed, behaviour is hard to generate in a way that remains independent from any technology. We are unaware of any existing approach building behaviour from the highest level (computing-independent model) to the lowest level (platform-specific model). Existing approaches only address some parts of some levels.

Concern#5: Lack of multiple platform managing

With the proliferation of platforms and accessories, it is not a luxury to use the same specification to develop an application usable in various environments. In other words, the need for generalization goes well with the need of specialization dialogue over several platform. Provide an environment that can manage multiple platforms is a challegence.

In this thesis, the challenge is to provide some answers that can help to alleviate these five shortcomings. Indeed, based on three pillars model, method and tool, we apply the paradigm of *Model-Driven Engineering*, MDE in short, to provide an integrated methodology of developping interactive dialogues. An assisted modelling approach in the specification, editing and / or generating code of an interactive application is offered to developers (designers, analysts, designers and / or programmers). According to Cameleon Framework Reference (CFR) i.e. whatever the level of abstraction (abstract, concrete or final), the methodology aims to provide concepts to achieve interactive dialogues with a model transformational approach.

To support the overall conceptual model of the methodology and prove its feasibility, we have implemented a graphical editor called Dialog Editor

Chapter 3 Model-Driven Engineering of Behaviours

We would have to insist from the beginning of this main chapter that our research describes a model-driven engineering of interactive dialogues in graphical user interfaces that is structured according to the three lowest levels of abstraction of the Cameleon Reference Framework: abstract, concrete and final user interface.

A dialogue model captures an abstraction of the dialogue as opposed to a traditional presentation model that captures the abstraction of the visual components of a user interface. The dialogue modelled at the abstract user interface level can be reified to the concrete user interface level by model-to-model transformation, which in turn leads to code by model-to-code generation.

This chapter is aimed at addressing the aforementioned challenges by applying MDE principles to designing a dialogue for GUIs belonging to different computing platforms. The remainder of this chapter is structured into three sections.

Section 1 describes our methodological approach. Initially, it sets the context by recalling some basic concepts. Afterwards, it treats MDE before showing how we integrate into our research these two notions. This section concludes with the presentation of the flowchart defining the algorithm to be followed while applying our methodology. This section concerns the Method branch in our methodological approach.

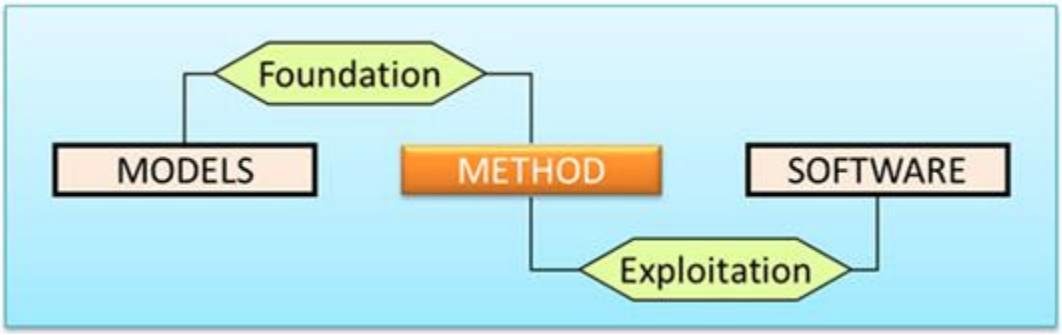


Figure 23. Method frame in Methodological diagram.

Section 2 examines the conceptual model. After listing and defining the useful elements in the dialogue specification, it establishes the links between these elements in the UML diagram of all object classes needed. According to the methodological approach, this chapter concerns the method pavement as shown in Figure 23.

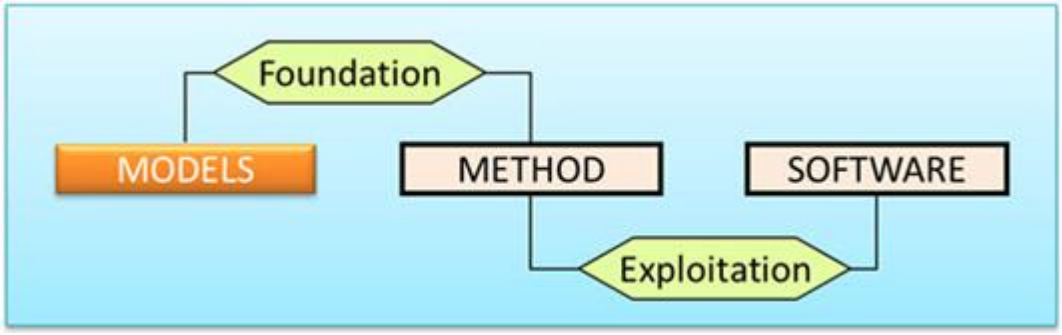


Figure 24. Models frame in Methodological diagram.

Section 3 focuses on software support in the *Dialog Editor* description. It motivates our software implementation with multi-level dialogue model editing, model-to-model transformation and model-to-code generation. It gives its technical characteristics and its conceptual and functional architecture. Here, we complete the methodological approach diagram by treating the practical aspects of our research.

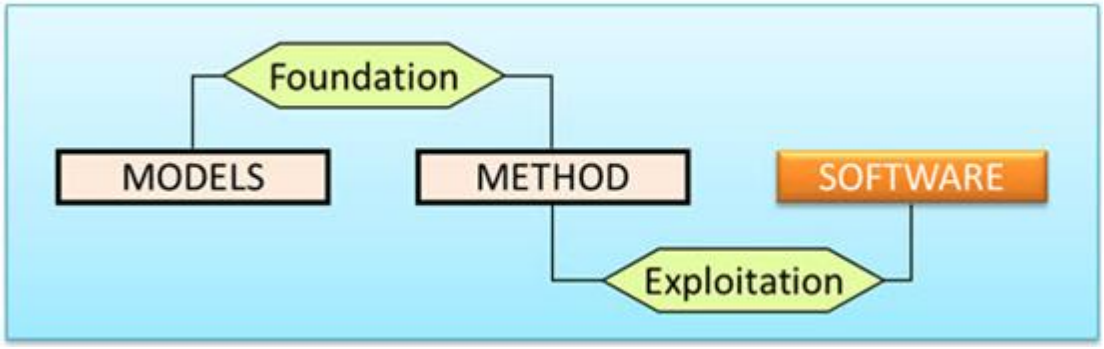


Figure 25. Software frame in Methodological diagram.

3.1 Methodology

By harnessing the principles and structure of *Cameleon Framework Reference* (CFR), the methodology we build is based on the *Model Driven Engineering* (MDE) approach. After the preliminaries of our research, this section aims to emphasize the fundamental concepts on CFR and MDE. Then it will describe our methodology using its flowchart.

3.1.1 Preliminary

We hereby refer to behaviour as being the dynamic part of a Graphical User Interface (GUI) such as the physical and temporal arrangement of widgets in their respective containers and their evolution over time depending on the user's task. Behaviour regulates the ordering of these widgets so as to reflect the constraints imposed by the user's task.

The dialogue has also been referred to as behaviour, navigation or feels (as opposed to look for presentation) [Ari88, Bas99, Elw96]. Here are some typical examples of dialogues: when the end user selected her/his native language in a list box, a dialogue box is translated accordingly; when a particular value has been entered in an edit field, other edit fields are deactivated because they are no longer needed; when a validation button is pressed, the window currently open is closed and another one is opened to pursue the dialogue. Conceptual modelling [Ari88], model-based design [Bre09] or model-driven engineering [Mei09a] of the dialogue was already introduced since years ago[Elw96] in order to be derived from a task model [Dit04,Luy03,Rei08,Van98,Van03], perhaps combined with a domain model [Tra03] or a service model [Bre09], to derive its software architecture from its model [PlI05], to analyze its properties [Cac07,Van99], to foster component re-use [Cow95], to check some dialogue or usability properties [Van99], to support adaptation [Men03], to automatically keep trace of interactions and analyze them afterwards [Rei08]. Dialogue models have been used in several domains of applications, such as web engineering [Boo07,Czc07], information systems [Mba03], multi-device environments [Sch07], multimedia applications [PlM05,PlI05], multimodal applications [Sch06] and workflow systems [Tra03,Tra08].

Dialogue modelling has however often been considered harmful for several reasons which may impedie further research and development in this area:

1. Choosing the modelling language paradigm is a dilemma: an imperative or procedural language is often more suitable and convenient to represent a GUI dialogue than a declarative language. The one could introduce a verbose representation of something that could be expressed in a straightforward way in the other. The current trend goes in favour of scripting languages.
2. Abstracting the right concepts is complex: finding the aspects of a dialogue that should lead to abstraction is not straightforward and turning them into an abstraction that is expressive enough without being verbose is difficult. A dialogue model may only have a limited level of expressiveness, but will prevent the designer from specifying complex dialogues while another dialogue model may exhibit more expressiveness, but is considered complex to use. Which modelling approach to use is also an open question: taking the greatest common denominator across languages (with the risk of limited expressiveness) or more (with the risk of non-support).
3. Heterogeneity of computing platforms is difficult to handle: Integrated Development Environments (IDEs) are often targeted to a particular programming language or markup language that is dedicated to a particular operating system or platform. Some IDEs exist (e.g. Nokia QT (http://qt.nokia.com/products, QtK) that address multi-platform GUIs, but they remain at the code level or their usage is still complex.
4. Model-driven engineering of dialogue is more challenging than model-based design. Model-based GUI design only assumes that one or many models are used to design parts or whole of a GUI, while Model-Driven Engineering (MDE) [Mei09] imposes at least one User Interface Description Language (UIDL) [Can10] that should be rigorously defined by a meta-model (preferably expressed in terms of MOF language, but not necessarily). Model-based GUI design may invoke virtually any technique, while model-driven engineering imposes the need for everything to be rigorously defined in terms of model transformations, which are in turn based on a metamodel.

We said above that the Methodology we propose is based on the Cameleon Framework. Before going any further, let us pause in the next section to analyse the founding principles of this environment.

3.1.2 Cameleon Reference Framework

Several UIDLs [Can10] are structured according to the four steps of the Cameleon Reference Framework (CRF) [Cal03], that are now recommended for consideration by W3C [Can10]:

- *Task & Concepts (T&C)*: describe the various users' tasks to be carried out and the domain-oriented concepts required by these tasks to be performed.
- *Abstract UI (AUI)*: defines abstract containers (AC) and individual components (AIC), two forms of Abstract Interaction Objects (AIO) by grouping subtasks according to various criteria (e.g. task model structural patterns, cognitive load analysis and semantic relationships identification). As in Guilet Dialogue Model [Rüc08] which enables flexible development with no restrictions on presentation and application layer and without any implementation-technology. The dialog model supports GUI designers and developers in understanding the behaviour of the GUI. One of the main keys is the independence to any interaction modality. The AUI is said to be independent of any interaction modality.
- *Concrete UI (CUI)*: concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. It abstracts a final UI into a UI definition that is independent of any computing platform. A CUI assumes that a chosen interaction modality, but the CUI remains independent of any platform.
- *Final UI (FUI)*: is the operational UI i.e. any UI running on a particular computing platform either by interpretation (e.g. through a Web browser) or by execution (e.g. after compilation of code in an IDE).

As noted already, our research refers to CFR. As the Figure 26 shows, we use the Moskitt XML schema and some UsiXML models. At the Final level, we aim for two environments Mac OSX and Windows for five programming languages; HTML V4.0, HTML for Applications (HTA), Microsoft Visual Basic for Applications V6.0 (VBA) and DotNet V3.5 framework

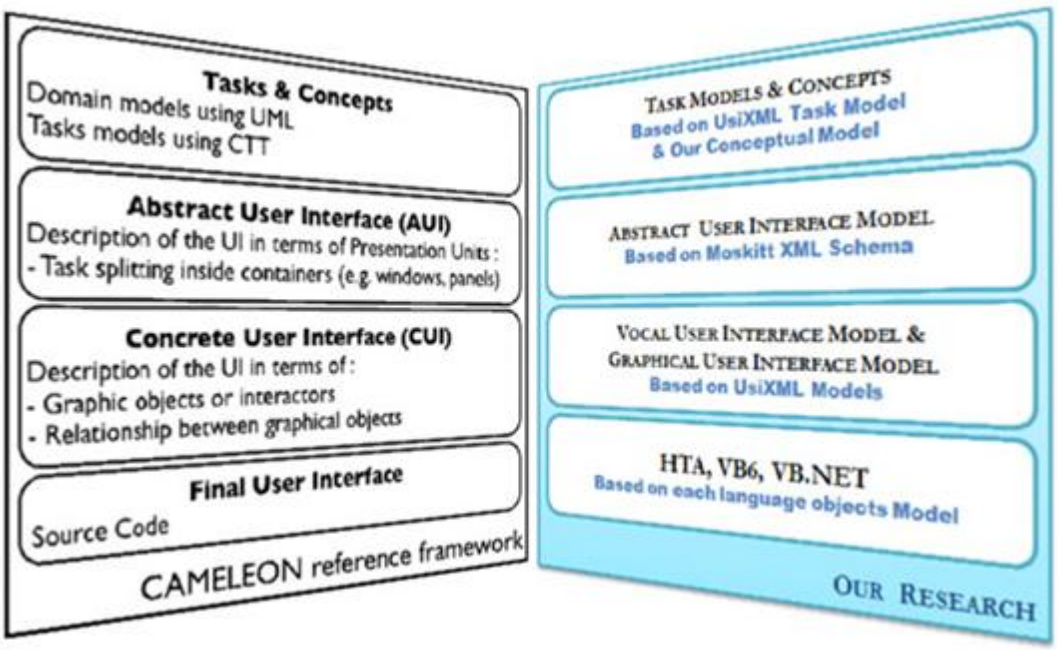


Figure 26. Application of CFR in our research.

3.1.3 Model-Driven Engineering

With the aim of proposing a methodology for specifying dialogues, we consider an interaction software applications as real things; things in real world. The real stuff is the code or the user interface objects, which are a collection of binary, text, graphical or formal documents the once put in a platform may run.

In this context, a model at final level of abstraction contains all required information regarding a specific platform that developers may use to implement the executable code. A concrete model describes the behaviour and structure of the application regardless of the implementation platform. An abstract model is the most abstract model which represents the context and purpose of the model without any computational complexities.

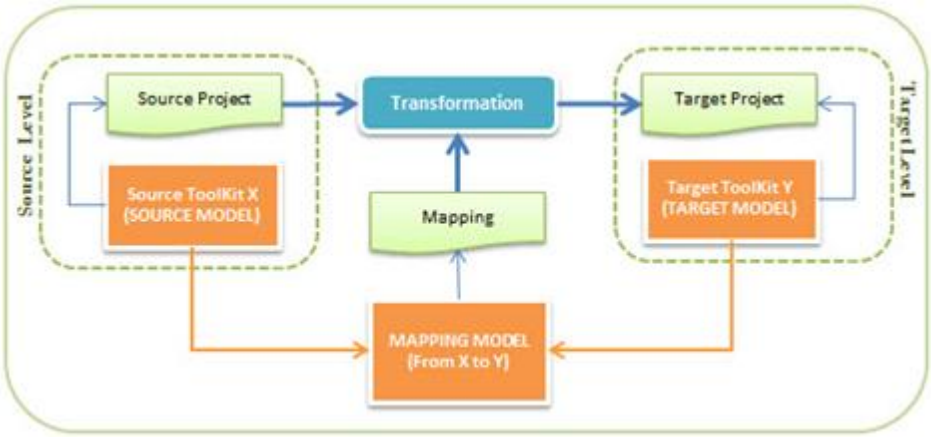


Figure 27. Applying MDE with Toolkits.

We have to insist on this confirmation, the main characteristic is that each exploited model is a toolkit; a box of objects whose syntactic and semantic properties furnish dialogue scripts. As shown is the Figure 27, Toolkits are classified according to the levels of abstraction of the Cameleon Reference Framework: task and domain, abstract user interface, concrete user interface and final user interface. The dialogue modelled at the abstract user interface level can be reified to the concrete user interface level by model-to-model transformation that can in turn lead to code by model-to-code generation.

The passage from a conceptual model to an actual application is accomplished through a succession of model transformations based on a Model-Driven Architecture (MDA). We applied these model transformations using Mappings from the point of view of Human–Computer Interaction (HCI).

Before continuing, we will insist that the Cameleon Reference Framework [Cal05] enables multiple development paths and not just *forward engineering*. In forward engineering, transformations are supposed to transform elements of a model into elements belonging to another model whose level of abstraction is inferior (this process is referred to as *reification*). In *reverse engineering*, transformations are supposed to transform elements of a model into elements belonging to another model whose level of abstraction is superior (this process is referred to as *abstraction*). In *lateral engineering*, transformations are applied on models belonging to the same level of abstraction, possibly the same one.

We recall once again, for our research, that five target markup and programming languages are supported: HTML V4.0, HTML for Applications (HTA), Microsoft Visual Basic for Applications V6.0 (VBA) and DotNet V3.5 framework. Two computing platforms support these languages: Microsoft Windows and Mac OS X. Five levels of dialogue granularity are considered: object-level (dialogue of a particular widget), low-level container (dialogue of any group box), intermediary-level container (dialogue at any non-terminal level of decomposition such as a dialogue box or a web page), intra-application level (application level dialogue) and inter-application level (dialogue across different interactive applications). The methodology we propose allows these three types of engineering:

(i) *Forward engineering*, where mappings transform successively the AUI model into a CUI model that, in turn, is transformed into a FUI for the fours following targets: HTML V4.0, HTML for Applications (HTA), Microsoft Visual Basic for ApplicationsV6.0 (VBA) and DotNet V3.5 framework. HTML V4.0 and HTA are running on both MS Windows and Mac OS X platforms.

(ii) *Reverse engineering*, where mappings transform something concrete into something abstract. Mapping for reverse engineering Visual Basic V6.0 code directly into an AUI model by establishing a correspondence between native objects and their corresponding user objects, two sub-classes of interactive objects.

(iii) *Lateral engineering*, where mappings transform model elements belonging to a same level of abstraction, but for another context of use. Before continuing, we must emphasize that our conceptual and technical choices are guided by a desire to easily integrate our results into the UsiXML environment. Indeed, conceptual model of dialogues has been implemented as UML V2.0 class diagram in Moskitt (www.moskitt.org) that gave rise to an XML Schema.

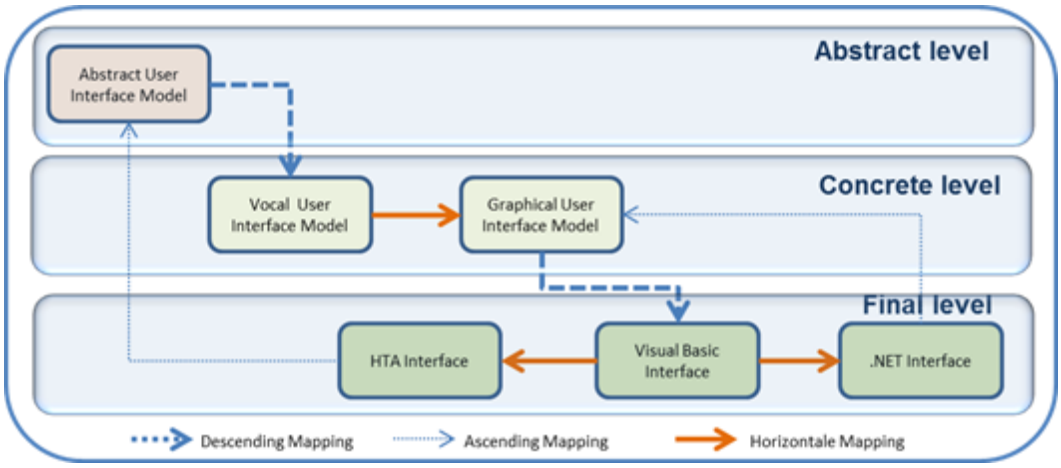


Figure 28. Three types of engineering in Contexte of Use.

As we can see in the Figure 28, it is very important to note that in this example, the reverse engineering does not need necessarily to work between two subsequent levels. Mapping can go from FUI directly to AUI without passing by the intermediary CUI level. This type of mapping is called *cross-cutting* as it represents a shortcut between two non-consecutive levels of abstraction. For example, a mapping for forward engineering from an AUI model directly to Visual Basic V6.0 code.

Referring to *Knowledge-Based Engineering* (KBE), we would like to add a useful extension. Indeed, a pool of knowledge or a database is added to the context of use. Such a database will aim to reduce time and cost of product development, which is primarily

achieved through automation of repetitive design tasks while capturing, retaining and re-using design knowledge[Ver12].

Therefore, developers will have to consult the online documentation for the models, features and all other concepts of the methodology. In addition, good practice scenarios will be added to support developers. They will find complete examples where each step will be described, and also simple illustrations as decisions support.

We have some regret not having formally treated this extension in the present state of our research. However, we have a series of video sequences, in French and English, illustrating the theoretical and practical elements of the methodology.

In addition, two documented examples are available. A small example of password management and a complete example of a CTI company. These two points are discussed later in the fourth chapter. In this way, the context of use can be represented by Figure 29 below.

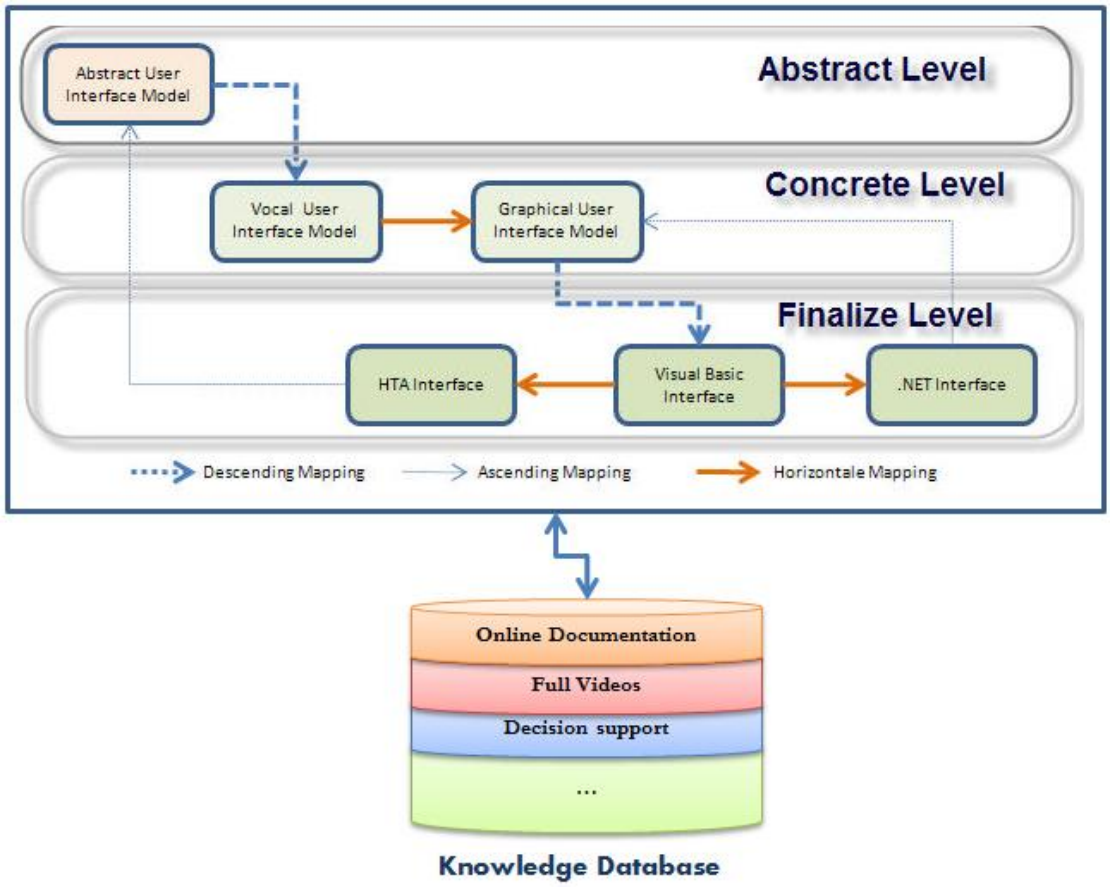


Figure 29. Extended Context of Use

Considering, firstly, the notion of model as a black box, the following section provides flowchart, the algorithm, of the methodology that we propose.

3.1.4 Applying the methodology

Based on the explanations of the previous section allow the Figure 30 gives the step-by-step instructions for applying the methodology that we propose. Indeed, the designer has to choose his/her own model and his initial level, following which he can create a new project or edit an existing project. After, by using mappings, he can move from one model to another. There are no limits, either for the model or for the level. That is one feature of this methodology. Additionally, if its model is final, he can choose to generate the code to be an executable.

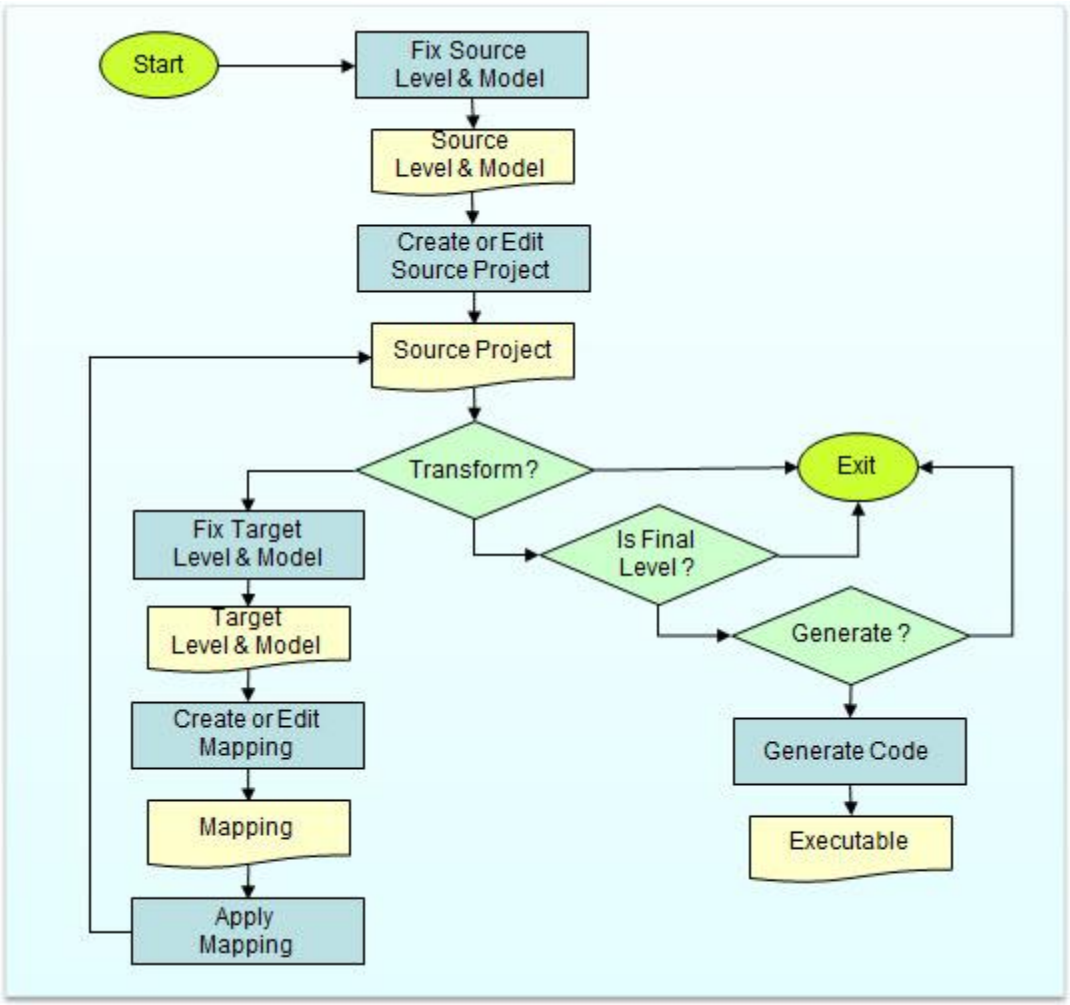


Figure 30. Methodology steps.

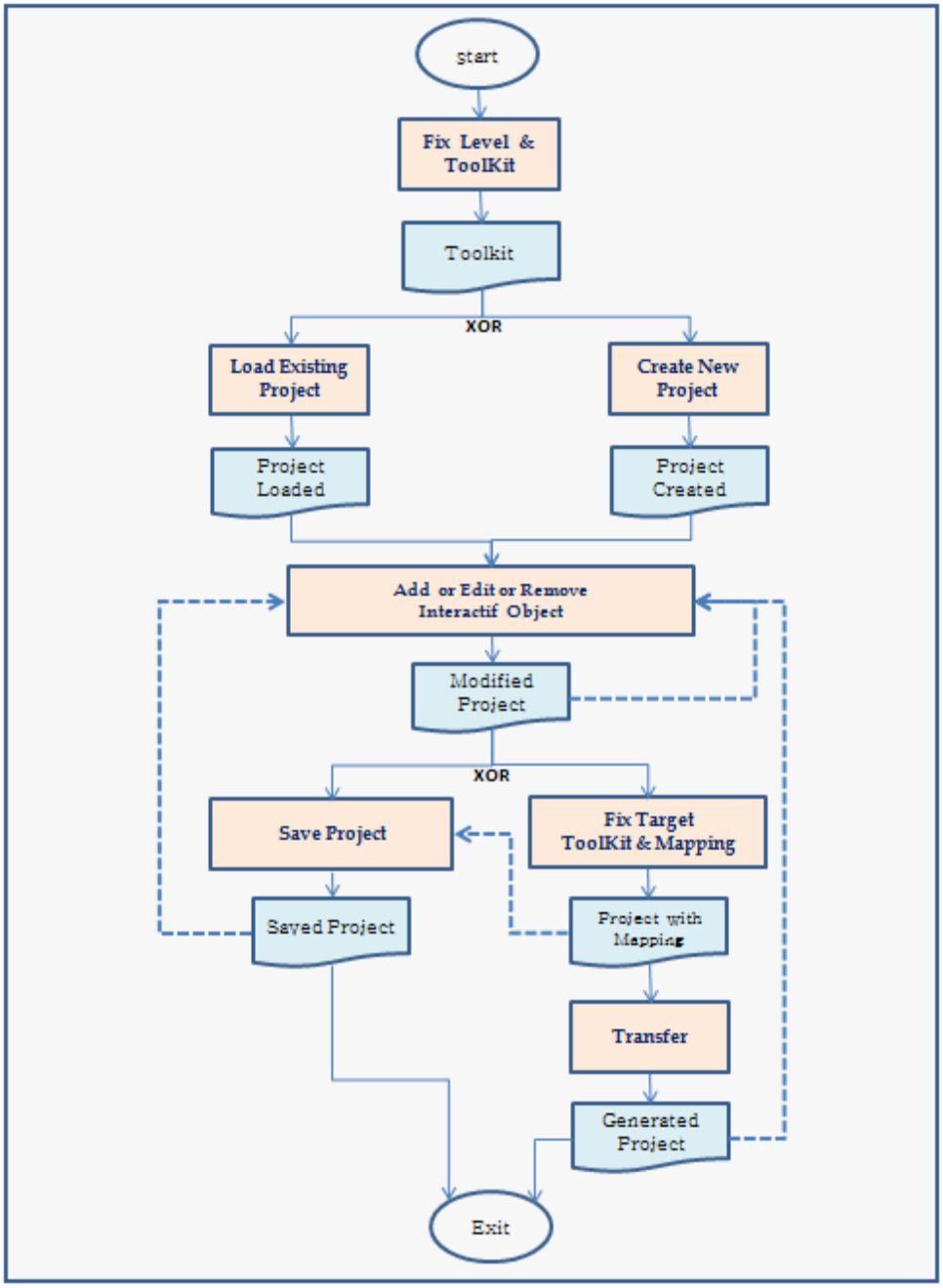


Figure 31. Project Editing Algorithm.

Take a second flowchart (Figure 31) that better illustrates some aspects of the methodology, in particular, the difference between creating a project and opening an existing project. Also, this new diagram emphasizes the addition or deletion of interactive objects in a given project. It is very interesting to note that a project created or specified in the context $<n_1,tk_1>$ i.e. at the level n_1 and the toolkit tk_1, can be transferred in the context $<n_2,tk_2>$ to be changed. The methodology allows multiple transformations before generating project codes from in a context $<n_m,tk_m>$ where n_m is the final level. The preceding remarks highlight the importance of clearly defining the mappings.

Indeed, if two mappings are very different, their applications to the same project can lead to different projects.

Now, open the black box to describe the more specific conceptual model of the methodology. This is the subject of the next section.

3.2 Conceptual Model

In order to apply MDE techniques, we need to define a dialogue model that is expressive enough to accommodate advanced dialogues at different levels of granularity and different levels of abstraction, while allowing some structured design and development of corresponding dialogue. The *Dialog Editor* described in this thesis will rely on this conceptual model. For this purpose, our conceptual modelling consists of expanding ECA rules towards dialogue scripting (or behaviour scripting) in a way that is independent of any platform.

3.2.1 Dialogue granularity

This dialogue scripting is structured according to a meta-model that is reproduced in Figure 32 that enables the definition of a dialogue at five levels of granularity:

1. *Object-level dialogue modelling*: this level models the dialogue at the level of any particular object, such as a CIO or an AIO. In most cases, UI toolkits and IDEs come with their own widget set with built-in, predefined dialogue that can be only be modified by overwriting the methods that define this dialogue. Only low-level toolkits allow the developer to redefine an entirely new dialogue for a particular widget, which is complex.

2. *Low-level container dialogue modelling*: this level models the dialogue at the level of any container of other objects that is a leaf node in the decomposition. Typically, this could be a terminal AC at the AUI level or a group box at the CUI level in case of a graphical interaction modality.

3. *Intermediary-level container dialogue modelling*: this level models the dialogue at the level of any nonterminal container of objects that is any container that is not a leaf node in the container decomposition. If the UI is graphical, this could be a dialogue box or the various tabs of a tabbed dialogue box.

4. *Intra-application dialogue modelling*: this level models the dialogue at the level of top containers within a same interactive application such as a web application or a web site. It therefore regulates the navigation between the various containers of a same application. For instance, the Open-Close pattern means that when a web page is closed, the next page in the transition is opened.

5. *Inter-applications dialogue modelling*: since the action term of an ECA rule could be either a method call or an application execution, it is possible to specify a same dialogue across several applications by calling an external program. Once the external program has been launched, the dialogue that is internal to this program (within-application dialogue) can be executed.

3.2.2 Interactive object

The current subsection introduces the concepts used towards the conceptual modelling of dialogues that could be structured according to the five aforementioned levels of granularity. These concepts are defined and motivated in the next sub-sections.

1. *Interactive Object*. An interactive object is the core component of the conceptual model as it consists of any object perceivable by the end user who could act on it. Interactive objects are further sub-divided into three levels of abstraction depending on the CRF [Cal03]: abstract, concrete and final (Figure 32 shows how this hierarchy is implemented in the *Dialog Editor* respectively at the three levels);

2. *Abstract Interactive Objects*. They describe interactive objects at the Abstract User Interface (AUI) level of the CRF. In the Dialog Editor, they are implemented as abstract classes compliant with Morfeo's Abstract UI model (http://forge.morfeo-project.org/wiki_en/index.php/Abstract_User_Interface_Model) which has been selected for the following reasons: Morfeo's AUI is one of the most recent efforts to define AUI that has been successfully implemented in the Morfeo project and has therefore been recommended as a reference model for the European NESSI platform (www.nessi.eu) through the FP7 Nexof-RA project (www.nexofra.eu) which promotes a reference software architecture for interactive systems, including the GUI part. Morfeo's AUI model holds two object types: an interactor manipulates data such as input, output, or both, through simple interaction mechanism (e.g. a selection) or through complex ones (e.g. a vector, a hierarchy); a container could contain interactors and/or other containers. Figure 32 details the definition of the abstract class implemented for the Free object that serves for general-purpose input/output;

3. *Concrete Interactive Objects*. They describe interactive objects at the Concrete User Interface (CUI) level of the CRF. Such concrete interactive objects may range from a simple widget such as a push button, a slider, a knob to more complex ones such as a group box, dialogue box, tabbed dialogue box.

   If we abstract an interactive object from its various physical representations that belong to the various computing platforms and window managers, any interactive object is characterized by its attributes and dialogue. An object may react to the end user's actions by handling events generated by this object.

   Therefore, a class could introduce an abstraction of object characteristics, including its attributes (fields or properties), its methods (through which a concrete interactive

object could be manipulated) and its events (that could be generated by, or received by, a concrete interactive object).

A class is hereby considered as a model of interactive objects of the same type. For example, a TextBox of a GUI consists of a rectangular widget for the entering of text, characterized by attributes including *width*, *height*, *backgroundColor*, *maxLength* or the *currentText*.

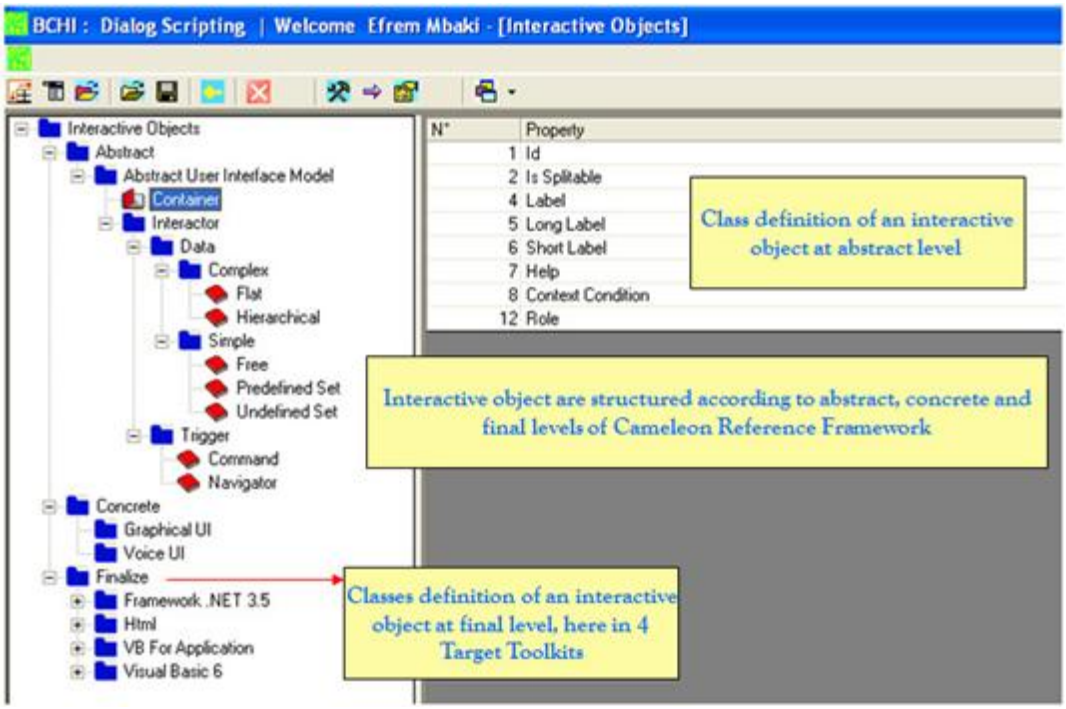


Figure 32. The hierarchy of interactive objects classes.

Textbox operators are also associated such as *appendText*, *giveFocus*, *selectAll* or *clearEntry*. A textbox generates events such as *textBoxSelected* when the textbox has been selected by any means (e.g. by clicking in it, by moving the tabulation until reaching the object) or *textBoxEnter* when the GUI pointer enters in the object (e.g. by moving the mouse-cursor into it or by touching it);

To fix ideas, let us make a forward reference to show how objects are organized in the editor. It should be emphasized that a section for a description of the editor is planned in the future.

3.2.3 Behaviour Model

This section is one of the most important of this thesis. Indeed, it describes the various objects of dialogue and data models between them. Our goal is to examine each of these objects under the microscope in order to understand what it brings to the specification of the dialogue. Based on the diagram below, let us define each entity:

1. *Final Interactive Objects*. They describe interactive objects at the Final User Interface (FUI) level of the CRF. In the Dialog Editor, they are implemented as real classes corresponding to the various toolkits supported (Figure 33 shows the four toolkits that are currently supported with the hierarchy expanded for Visual Basic V6.0). For each interactive object, only the common native dialogue is factored out and rendered as a sub-class of the toolkit. This is why final interactive objects are represented as native objects in Figure 33, while abstract and concrete interactive objects are represented as user-defined classes in Figure 33. We hereby assume that the native dialogue of any final interactive object is preserved. For defining non-native dialogues of a final interactive object, dedicated methods exist, such as the Interaction Object Graph (IOG) [Car06]. Since defining custom dialogue at the control level requires complex and dedicated programming, it is not supported unless such a dialogue can be characterized as an interactive object.

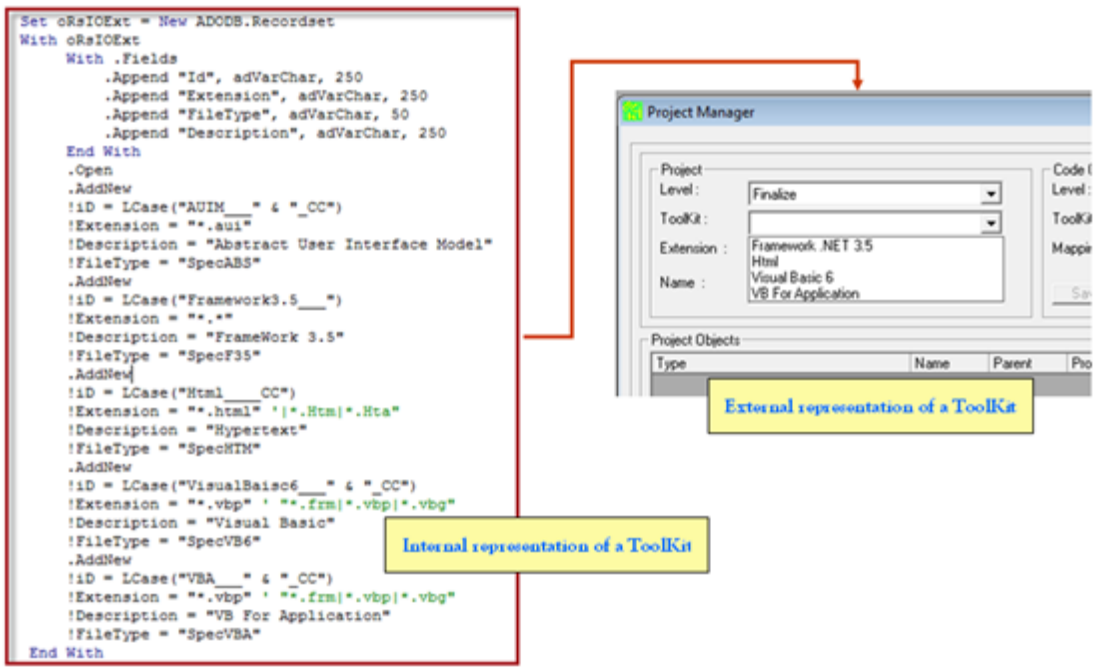


Figure 33. Internal and external representation of Toolkits.

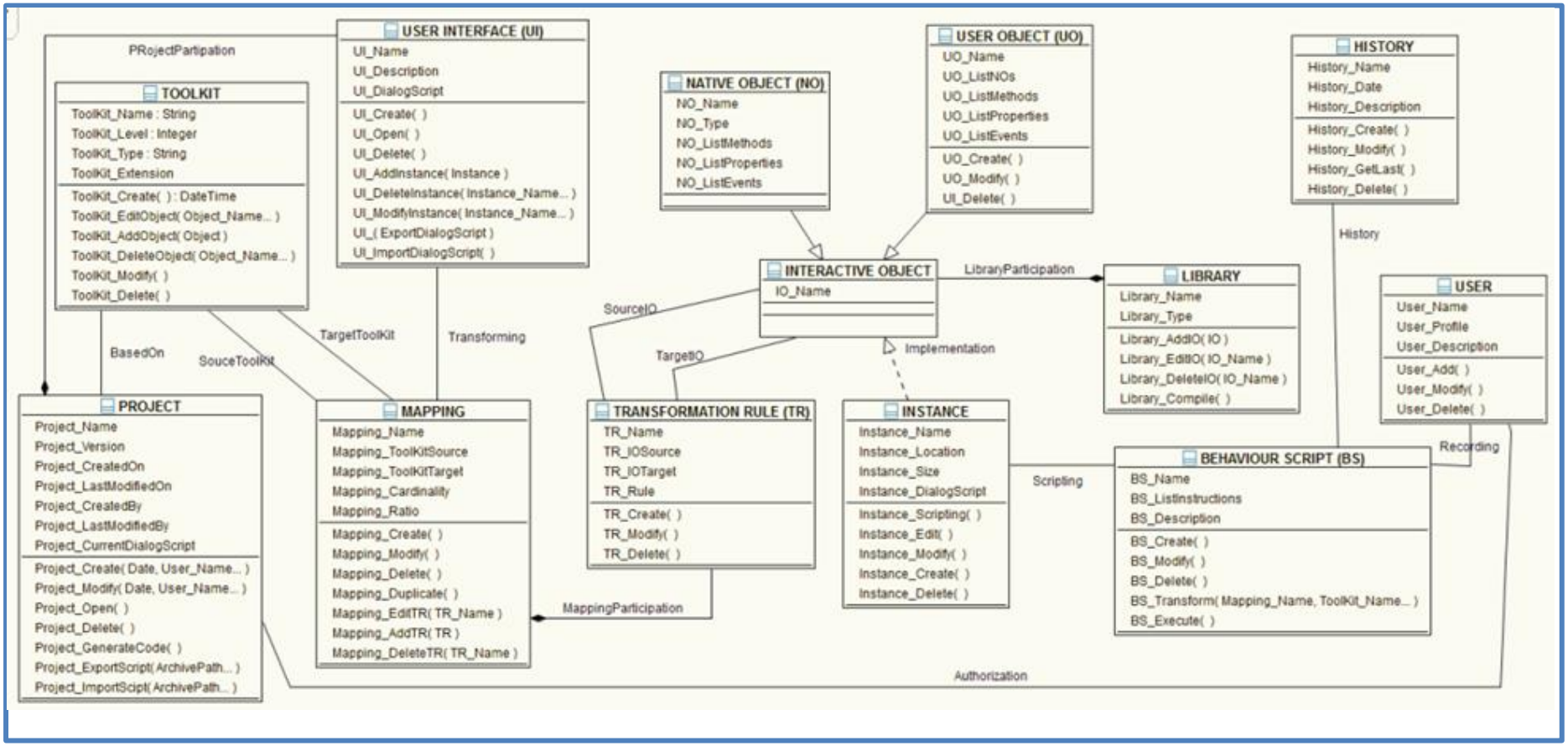


Figure 34. Conceptual modelling of behaviours for model-driven engineering.

Now that these five levels have been defined, we introduce the concepts used towards the conceptual modelling of behaviours that could be structured according to the five aforementioned levels of granularity. Figure 34 depicts the global structuring of these concepts:

- ***Toolkit***: In order to support GUIs for multiple computing platforms, each supported toolkit of a particular platform is characterized by its name, its level (e.g. a version), its extensions and a series of templates describing how this toolkit implements particular dialogues. Three values are accepted depending on abstraction level: abstracted (AUI), concrete (CUI) or final (FUI). Figure 34 shows the correspondence of the external representation of a toolkit that is visible to the end user and the representation inside Dialog Editor.

- ***Library***: A library gathers a series of particular interactive objects at any level so as to refer to them as a whole, which is helpful for keeping the same definitions for one target computing platform, typically a toolkit. For the moment, HTML V4.0 is one of the toolkits supported by its corresponding library. Any newer version of HTML, e.g. V5.0, requires implementing a new library for this toolkit.

- ***Instance***: An instance is any individual object created as an instance of any interactive object class. While a class defines the type of an interactive object, any actual usage of this class is called "instance". Each class instance possesses different values for its attributes. At any time, the instance state is defined by the set of its attributes values. By respecting the encapsulation i.e. the process of hiding all the attributes of an object from any outside direct modification, object methods can be used to change an instance state. In order to have a login+password, two instances should be created that share the same definition, but with different instance states.

- ***User Interface***: A User Interface (UI) as it is considered in this conceptual model may consist of any UI at any level of abstract (i.e. abstract, concrete, or final). Therefore, such a UI consists of a set of instances each belonging to the corresponding level of abstraction.

- ***Project***:: A project is considered as a set of UIs for a same case study for a particular toolkit. In a same project, one can typically find one AUI, one CUI and one FUI. Of course, for the same AUI, different CUIs could be created that, in turn, lead to their corresponding FUIs. Actually, a project could hold as many CUIs and FUIs as model-driven engineering has been applied to the same AUI. This is achieved through the mechanism of mapping.

- ***Mapping***: In order to support model-driven engineering, a mapping is hereby referred to as any set of transformation rules from one source toolkit to a target toolkit. Note that source and target toolkits could be identical. A transformation rule is written as a PERL regular expression applied from a source class of interactive objects to a target class of interactive objects. In order to support Model-to-Model (M2M) transformation, a transformation rule may be applied

from one or many classes of abstract interactive objects to one or many classes of concrete interactive objects. For Model-to-Code (M2C) generation, a transformation rule is applied from one or many classes of concrete interactive objects to one or many classes of final interactive objects (so-called native objects). Let us consider again the login and the password example. At the abstract level, two instances of entry fields are created to be mapped onto objects belonging to a particular toolkit. In HTML, both fields are transformed into Input objects, respectively of type Text and Password. In VB6, they are transformed into two text boxes. For the password, IsPassword is set to True.

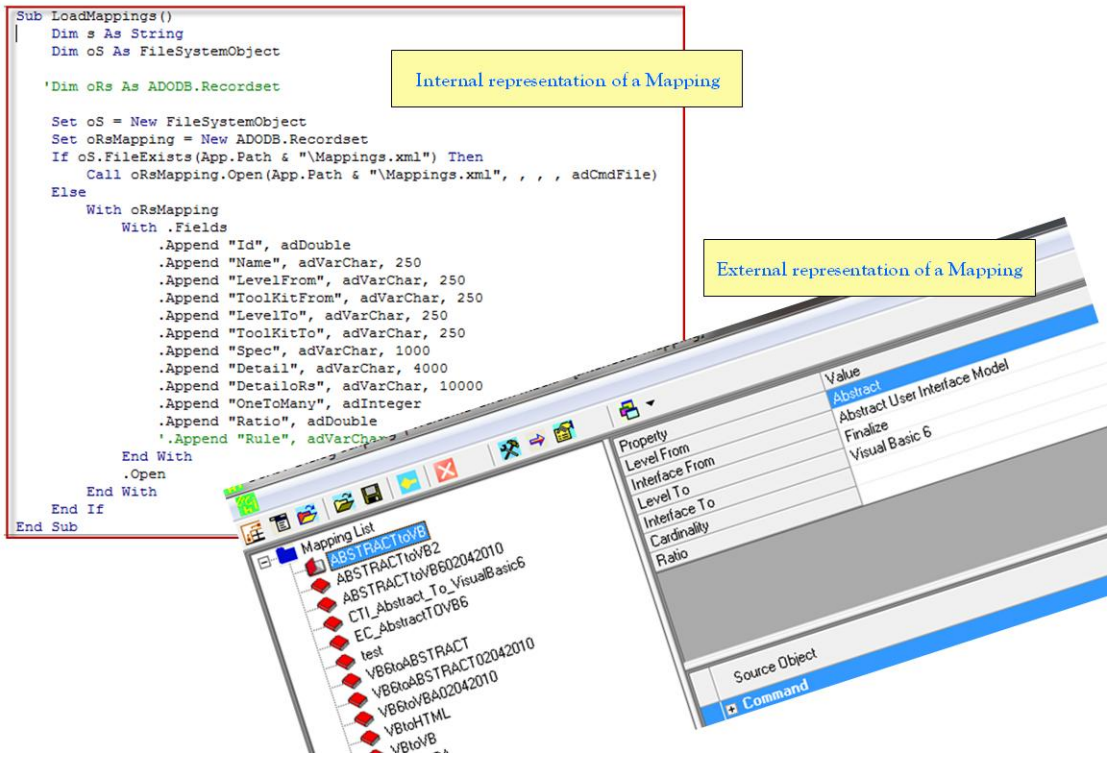


Figure 35. Internal and external representation of mappings.

Figure 35 shows both, the external representation of a mapping that is visible to the end user and its internal representation inside the Dialog Editor. Note that the Cameleon Reference Framework [Cal03] enables multiple development paths and not just forward engineering. In forward engineering, transformations are supposed to transform elements of a model into elements belonging to another model whose level of abstraction is inferior (this process is referred to as reification). In reverse engineering, transformations are supposed to transform elements of a model into elements belonging to another model whose level of abstraction is superior (this process is referred to as abstraction). In lateral engineering, transformations are applied on models belonging to the same level of abstraction, possibly the same one. Mappings as supported by the *Dialog Editor* support the three types of engineering:

(1) Forward engineering, where mappings transform successively the AUI model into a CUI model that, in turn, is transformed into an FUI for the four following targets: HTML V4.0, HTML for Applications (HTA), Microsoft Visual Basic for

Applications V6.0 (VBA) and DotNet V3.5 framework. HTML V4.0 and HTA are running on both MS Windows and Mac OS X platforms.

(2) Reverse engineering, where mappings transform something concrete into something abstract. Figure 36 depicts a mapping for reverse engineering Visual Basic V6.0 code directly into an AUI model by establishing a correspondence between native objects and their corresponding user objects, two sub-classes of interactive objects.

(3) Lateral engineering, where mappings transform model elements belonging to a same level of abstraction, but for another context of use. Before continuing, we must emphasize that our conceptual and technical choices are guided by a desire to easily integrate our results into the UsiXML environment. Indeed, the conceptual model of dialogues has been implemented as UML V2.0 class diagram in Moskitt (www.moskitt.org) (Figure 36) that gave rise to a XML Schema.

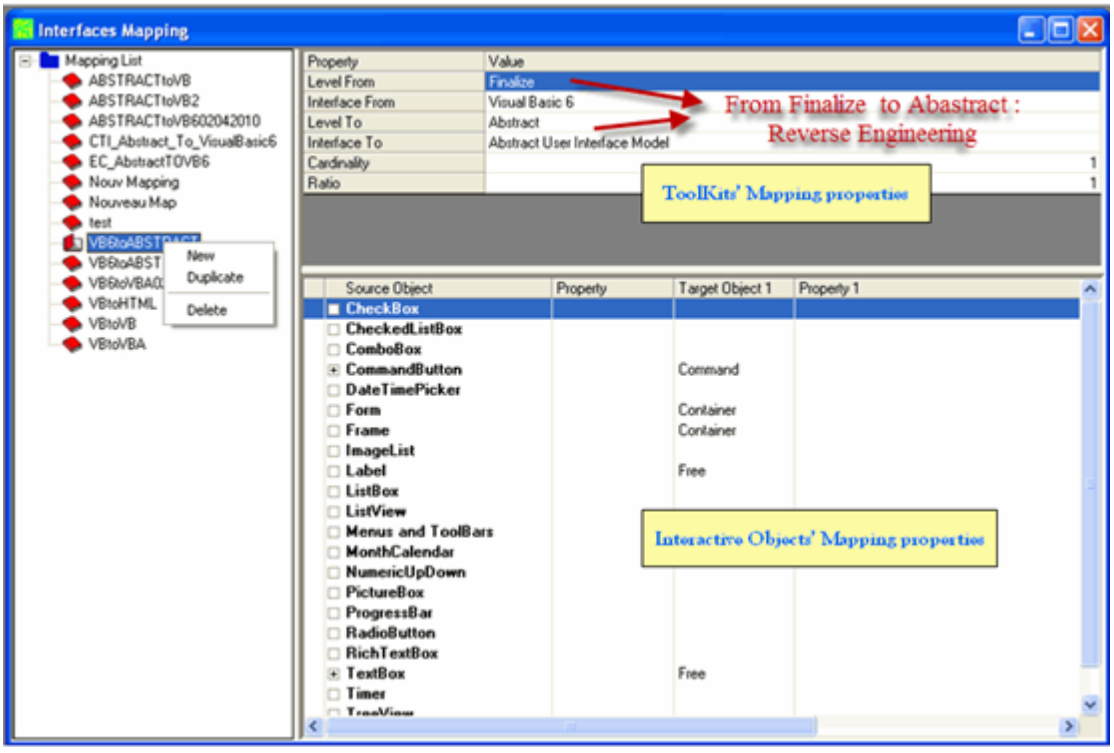


Figure 36. Example of a mapping for reverse engineering.

Note also that in this example, the reverse engineering does not necessarily need to work between two subsequent levels. The mapping depicted in Figure 36 goes from FUI directly to AUI without passing by the intermediary CUI level. This type of mapping is called cross-cutting as it represents a shortcut between two non-consecutive levels of abstraction. For example, Figure 37 depicts a mapping for forward engineering from an AUI model directly to Visual Basic V6.0 code.

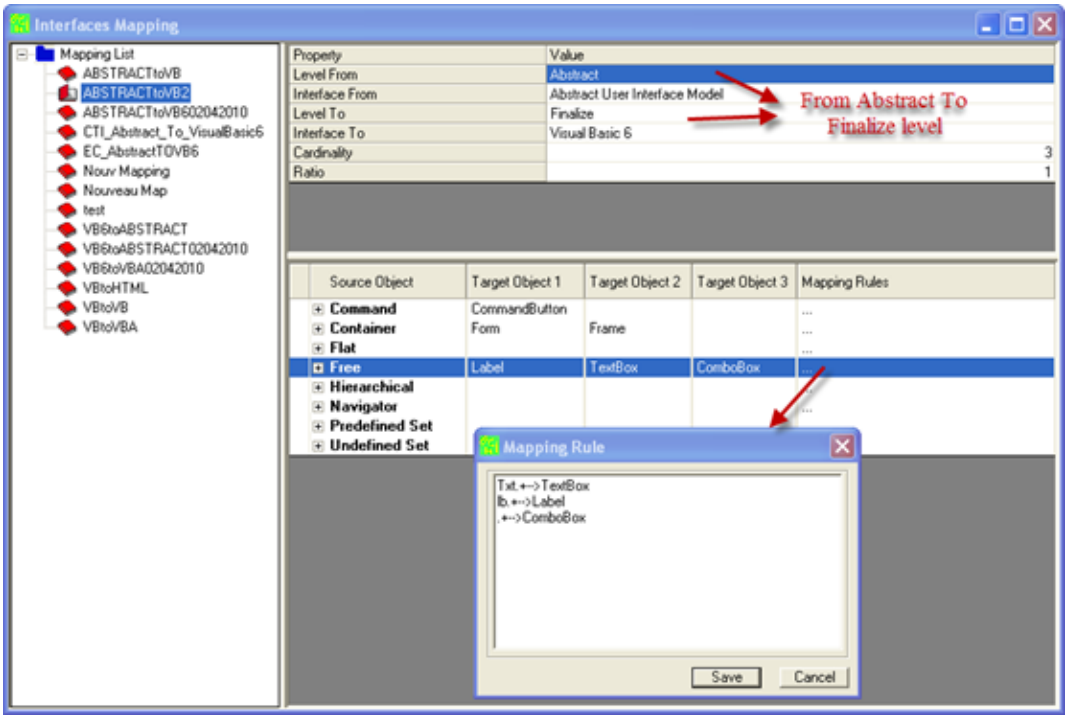


Figure 37. Example of "one-to-many" mapping.

In the Cameleon Reference Framework, multi-target is also described in terms of different contexts of use. Therefore, any mapping that goes from one context of use to another one is referred to as lateral engineering. The Dialog Editor also supports this through mappings at the same level of abstraction, but across two different contexts of use, such as between VB6 and HTML V4.0 (Figure 38).

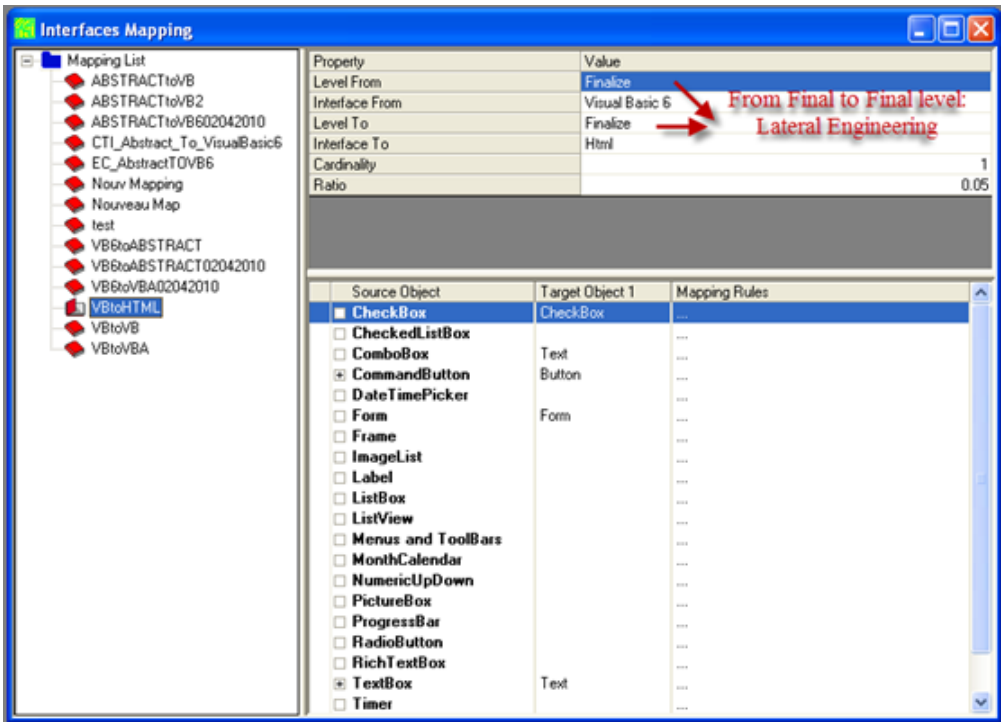


Figure 38. Example of mapping for lateral engineering.

- **Dialogue Script**: A dialogue script (or behaviour script) is a sequential text expressing the logical and conditional elements. It describes the actions to be achieved according to a given interaction scenario. An action can be the change of an attribute value, the call of a semantic function belonging to the functional core, or the opening or the closing of another user interface. Three levels of script are possible:

  1. Elementary dialogue scripts. These scripts are related to instances found in a given project. Often, these scripts are systematically generated accordingly to a template-based approach. They can come from:

     - A change of an attribute value: for example, a read-only field implies automatic database requests in its dialogue script ;
     - A layout positioning: for example, two interactive objects may be laid out in their parent according to an adaptation mechanism.

  2. User interface Scripts. These scripts relate to the implicit or explicit data exchanges between two or several interactive components having a common interactive ancestor. For example, an interactive object is activated or deactivated depending on the state of another object. The verification of a login+password can be initiated only after both fields are properly filled in.

  3. Project scripts. These scripts express the data exchanges between two or more interactive objects that are independent as they do not share any parent.

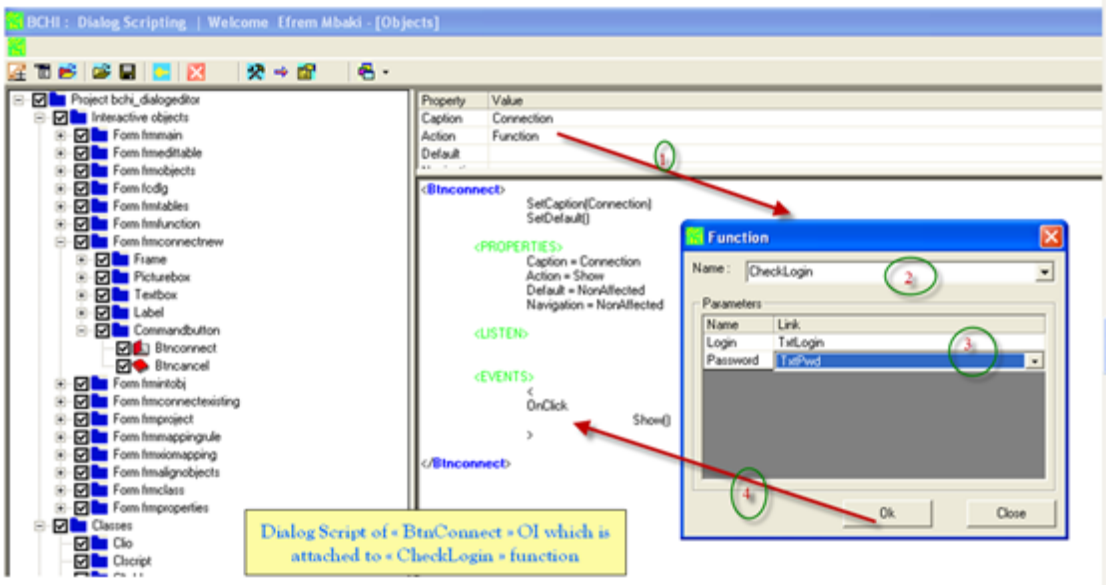


Figure 39. Dialogue script of an interactive object.

Any dialogue script is structured into three parts (Figure 39): a condition of realization, the event to consider and a list of actions to be undertaken when the event is fired and the condition is satisfied. A single script language has been defined in common for all

three types of dialogue scripts. These scripts use in harmony three models of dialogue; transition networks, grammars and events [Gre86]. Scripts of dialogues at the abstract and concrete levels are written using a generic language that we described using a BNF grammar. At final level, the code generator translates from generic scripting to specific language relative to a target model. It should be noted that some of these scripts are automatically deducted through some attribute values. A simple example is to associate the exit of an interactive task with the click of a button. Such a script is generated automatically. As in useML editor [Mei09], other scripts are derived semi-automatically. Indeed, by combining the event of an interactive object to a function call, the developer will have to make the links between the function parameters (input and output) with the attributes of interactive objects. Then, the editor automatically builds the script.

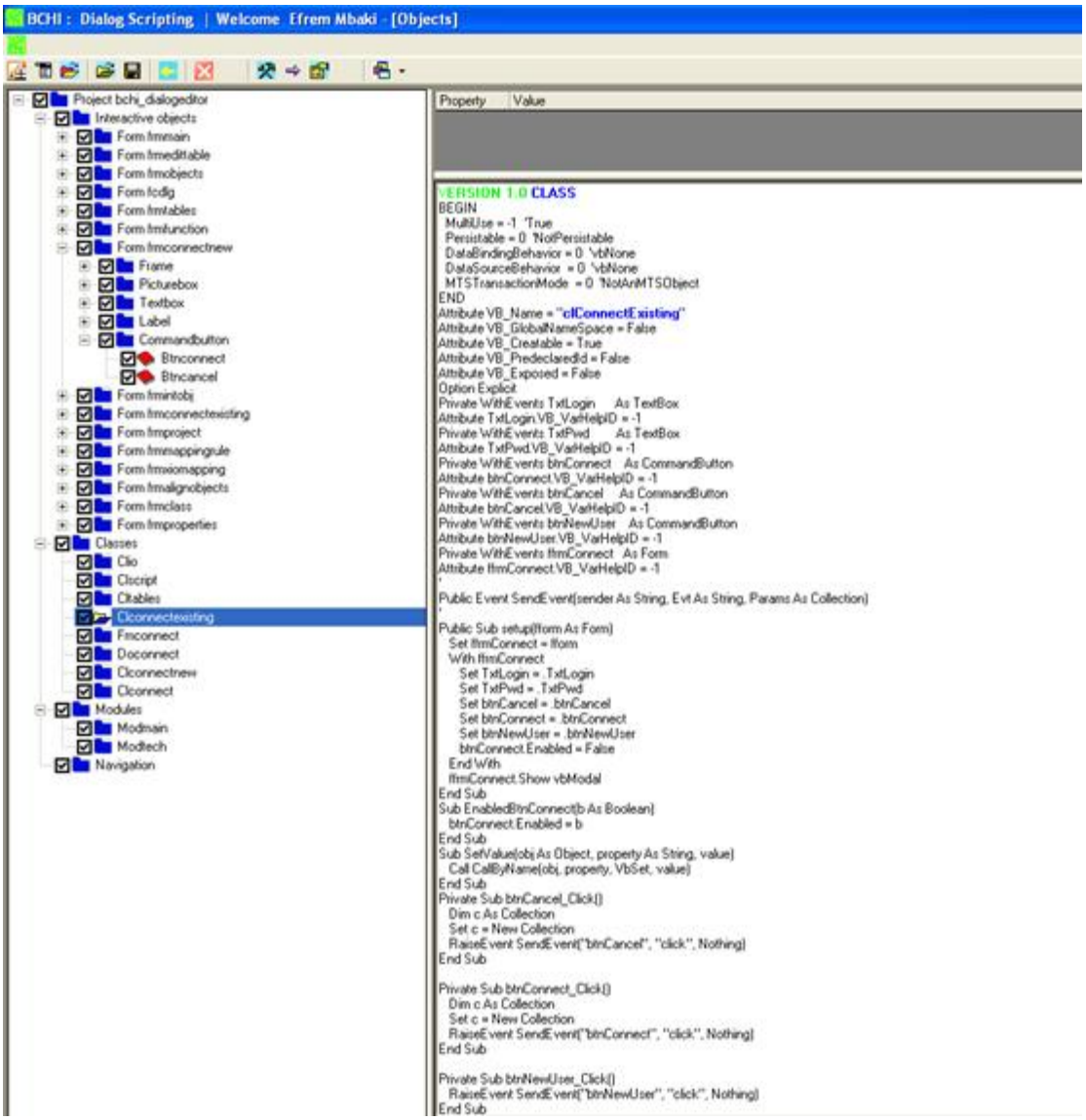


Figure 40. Script of Connection Class In Dialog Editor.

- **History**: A history consists of a set of time-stamped operations applied to dialogue scripts over time in order to preserve the design history. In this way,

some traceability of dialogue scripts (i.e. who created, retrieved, updated, deleted which dialogue script over time in the same project) and some reusability (i.e. copy/paste an already existing dialogue script) are ensured (Figure 40). Any dialogue script definition can be validated for a particular toolkit.

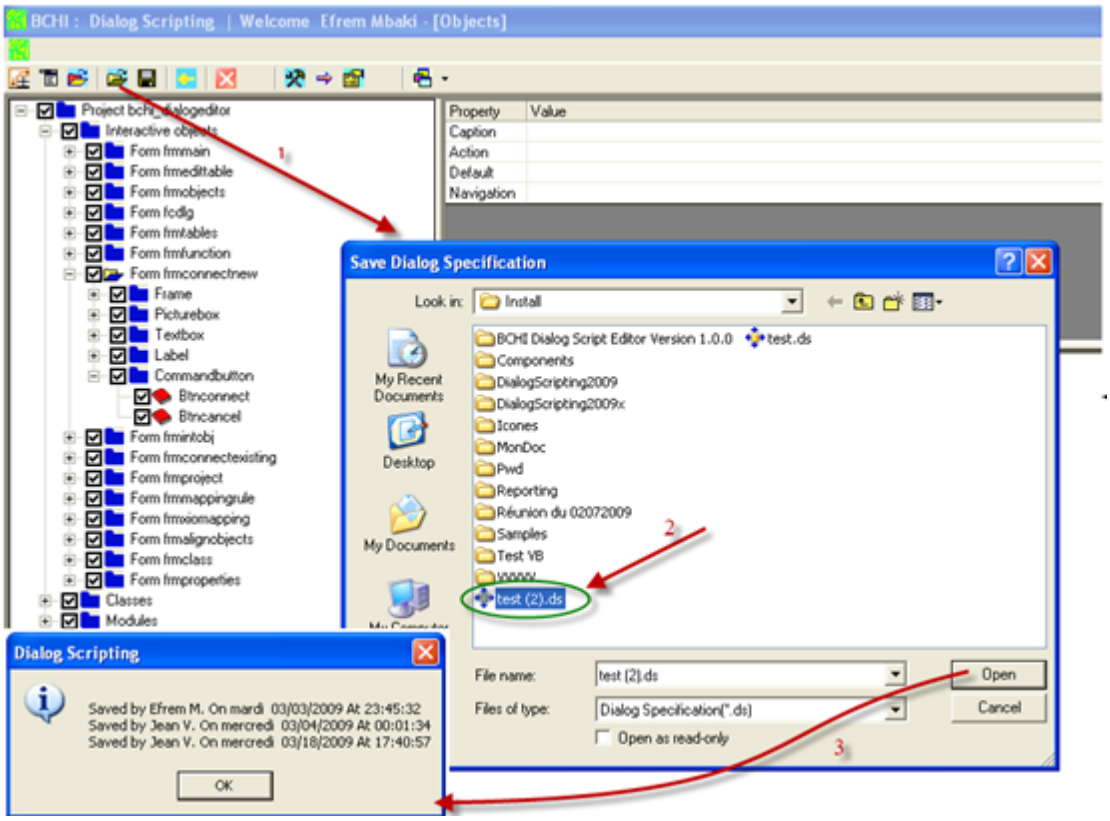


Figure 41. Recovering a previously saved history.

3.3 Implementation

To support the model and process described in the previous sections, we implement a graphic *Dialog Editor* in which Models are organized in three levels (abstract, concrete and final) according to CFR and, whose process respects the MDE approach.

The purpose of this section is to describe this behaviour Editor and present the technical choices we made. We should clarify that this is not a production tool. As part of our research, the purpose of this tool is twofold;

1. show the harmonic relationship between theoretical concepts and;
2. illustrate the feasibility of the methodology and the proposed algorithm.

As shown in the flowchart of Figure 30, the global algorithm processes by steps and respects mapping to move from model to another.

3.3.1 Software architecture

In order to achieve the goal of Model-Driven Engineering of dialogues for multi-platform GUIs based on the conceptual model of Figure 34, the process of the flowchart of Figure 30 and Figure 31 is decomposed into four main phases (Figure 42):

(i) Project editing which includes all facilities required to create, retrieve, update and delete any UI project during the development life cycle;

(ii) Project transforming which is aimed at supporting the creation of new mappings between levels and applying them via a mapping editor;

(iii) Scripting which is aimed at specifying any desired dialogue script at any time, before or after transformation;

(iv) Code generating which calls the mappings corresponding to the target platform for which the code of the FUI should be produced.

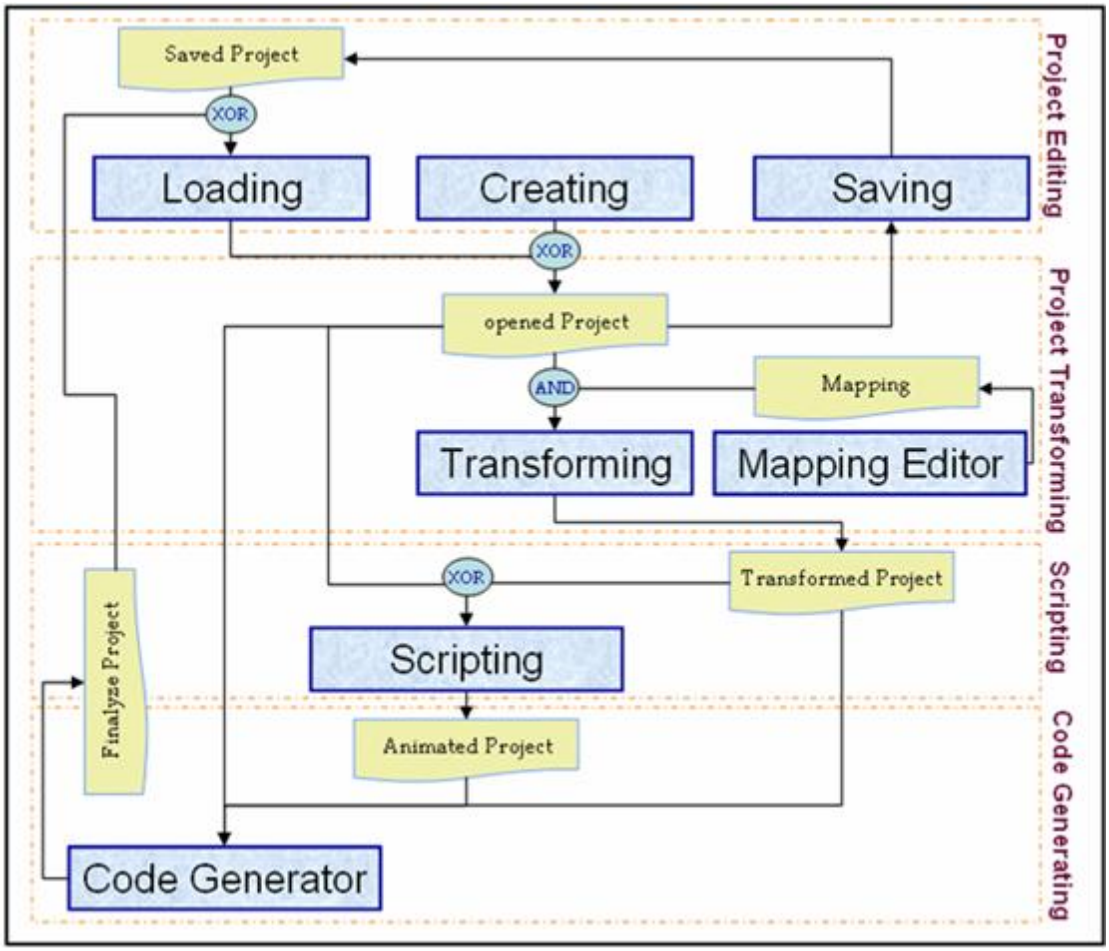


Figure 42. Dialog Editor Architecture.

Functionally, *Dialog Editor* offers all the services of project management; creation, editing or deleting. The Control mapping is one of the major modules of the tools. Code generation requires that the project be in final level. Otherwise, it is mandatory to use a mapping whose target level is final. Figure 43 provides a functional overview of the tool.

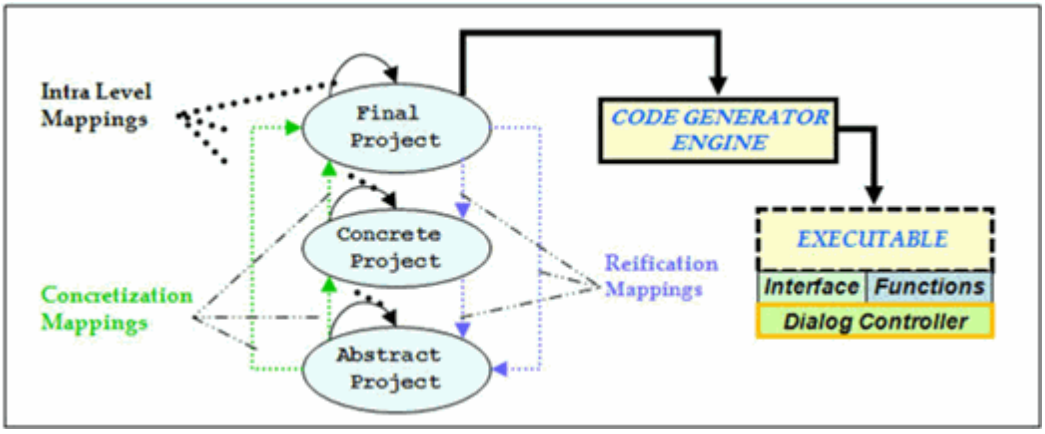


Figure 43. Dialog Editor functional overview.

3.3.2 Programming

The Behaviour Editor has been entirely programmed with Visual Studio 6 (*VB6*) and Visual Basic for Applications (*VBA*).

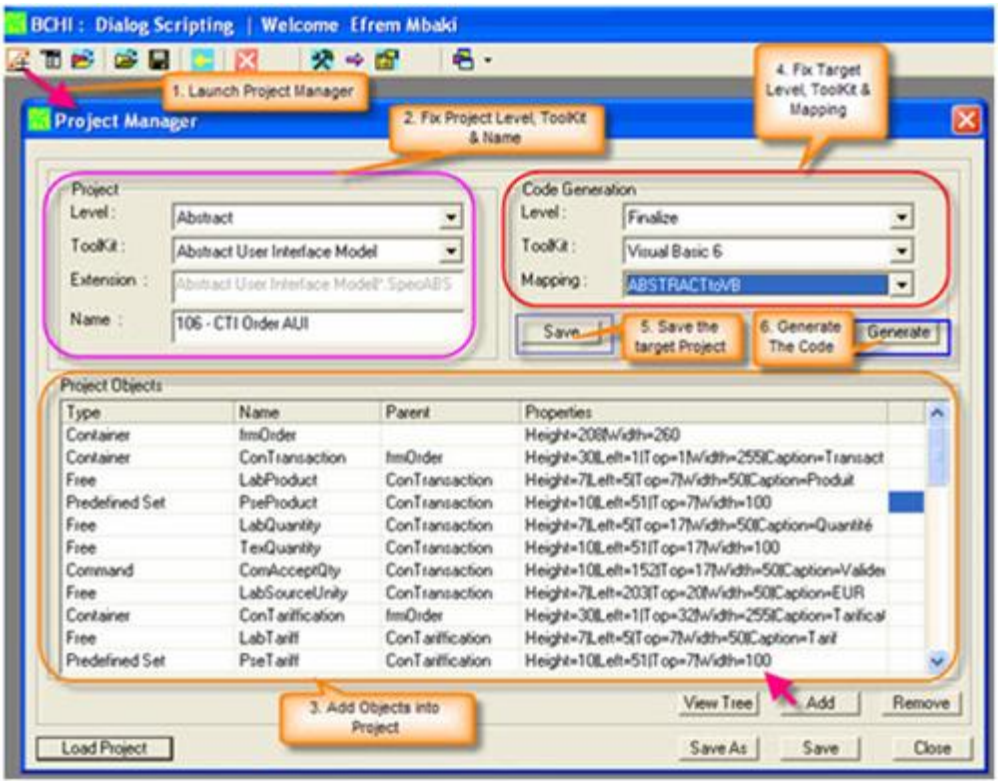


Figure 44. Project main window in Dialog Editor.

To standardize the presentation (look) and the dialogue exchange (feel), an ActiveX control has been developed. We have put in this ocx component known objects such as plain text (*textbox*), the simple list (*combox*), check (*checkbox*) or editable grid to manage tables.

Description	URL
Global View	**http://www.youtube.com/watch?v=x3CtCj47iZQ**
Architecture	**http://www.youtube.com/watch?v=Nx3d-w19Oug**
Project Editing	**http://www.youtube.com/watch?v=GRKWwq5cQzU**
Mappings	**http://www.youtube.com/watch?v=gVQ8bz9wEXY**
Scripting	**http://www.youtube.com/watch?v=EZGtL7fXtlE**
Code Generation	**http://www.youtube.com/watch?v=n7YlgpDihtY**

Figure 45. Video demonstrations of the *Dialog Editor.*

The table above (Figure 45) gives some references to videos we have made and published on YouTube to illustrate the use of the Dialog Editor. Each of these videos shows one of the modules of the architecture presented above.

In VB6, ActiveX Data Object (ADO) *Recordset* object is used to hold a set of records from a database table. Recordset object consists of records and columns (fields). As shown in Figure 46, we choise to use Recordsets for the internal organization of all components handled in the editor.

Moreover, VB6 offers a function used in saving a *Recordset* as an XML file. There exists also a function which opens a *Recordset* from an XML file. The Recordset's save method writes extended schema information to the XML file; this information is required to open an XML file into a recordset. Figure 47 shows an example of a saved file.

Figure 46. A Recordset for native objects.

Thus, we confirm that UIDL which is maintained by the *Dialog Editor* is therefore based on XML Schema. Therefore, any project created in the editor is compliant with the XML Schema.

Figure 47. XML file corresponding to a UI Project.

3.3.3 Script Editor

The *Dialog Editor* that we implemented includes a powerful module, *Script Editor*, in which the developer specifies the object's properties and/or script. For "Projet1", Figure 48 shows the tree of its objects and the characteristics of "*command1*", one of its objects.

We notice that the script of an interactive object consists of three parts:

1. **Properties**:

Depending on requirements, a list of properties of the current object is proposed. The developer can fill some of these properties with respect to its interface or to the dialogue he wants.

For example, for the object "*command1*" in Figure 48, we fill the label "quit" and we say that the object is associated with the action *Exit*. Also, we specify that this object should have the main focus among the offspring of its parent.

We will see in the third part, the impact each of these properties impact on the dialogue script of the object.

2. **Listening objects**:

In this section we fix the list of objects that could send an event to the current object. The objective is to provide treatment at the reception of any other event. In the example above, the object *command1*, will not listen.

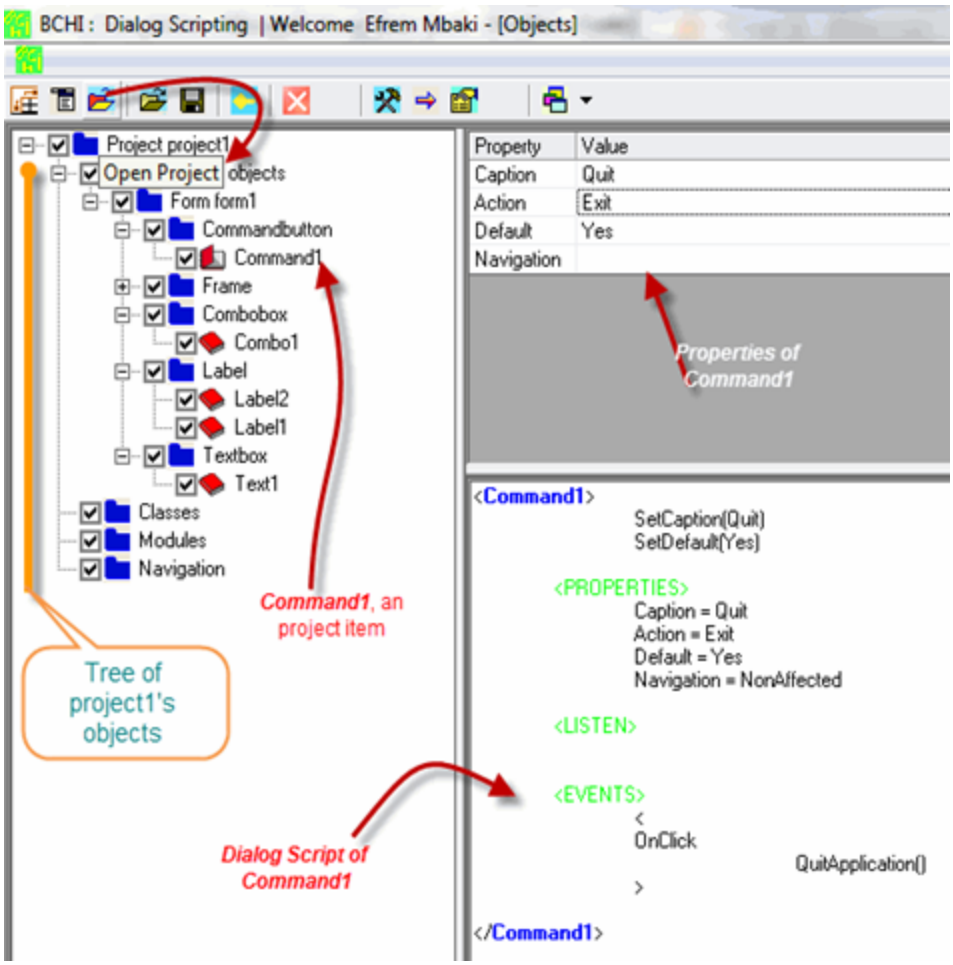


Figure 48. Script Editor.

3. **Events process**:

This is the part of the script itself. The developer will describe the instructions to be executed upon the occurrence of an event. Thus, each treatment begins with **OnEvent_Name.**

For the example above, we are in VB6 and we handle a GUI. The click of *command1* evokes the exit function. This call corresponds to the value assigned to the property *'action'*. Moreover, this script is automatically generated.

Here, the developer has the freedom to use its library functions. For that it should predict the translation of these functions into the target language.

We limited ourselves to defining the overall structure of the syntactic components without going into details. In other terms, we use a kind of pseudo language without semantics.

3.3.4 Mapping Editor

Mapping is one of the fundamental concepts of our research. We are constantly writing since the beginning of the document that our methodology is cross-platform and, to pass from one platform to another, we use a Mapping.

Indeed, cross-platform software has the advantage of being operational in several platforms i.e. they are compatibles for several couples binding computer and operating system. Nevertheless, for technical and/or ergonomic reasons, it frequently happens that the passage from one platform to another imposes certain transformations, particularly for the user interface (UI).

A platform describes architecture and framework that allows software to run. In detail, typical platforms include a computer's architecture, operating system, programming languages and related user interface (runtime libraries or graphical user interface computing). Each platform has its own characteristics such as the device's form factor, the appropriate interaction metaphors and the supported user interface toolkit.

A Transformation Rule (TR) is a mechanism which defines methods to replace one interactive object by another. Let us recall that Interactive Objects (IO) are particular objects used in the design and/or the implementation of User Interface (UI). Like any other object, an IO is characterized by its own properties, methods and events. For finalized objects, this includes forms and controls. Properties can be thought of as the attributes of an object, methods as its actions and events as its responses.

That being so, independently of the platform, to transform an object supposes the taking into account, at least, of source properties, methods and events versus target properties, methods and events.

TRs are an effective means of exceeding the differences between platforms by offering alternatives which project an image of a given UI from one platform to another. A TR receives an IO as an Input. Each IO is described by its attributes, its methods and its events. As output, a TR produces another IO, possibly the same one, which is an "image" of the input IO.

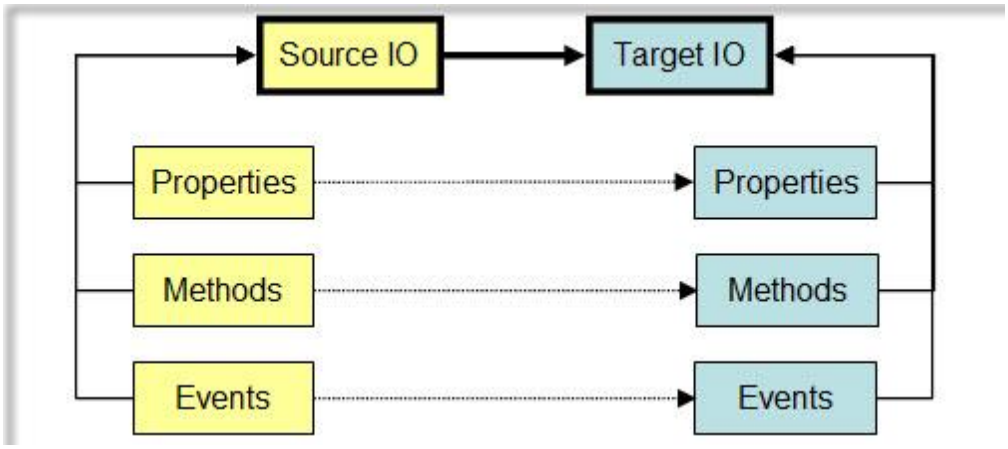


Figure 49. Objects Mapping.

In this context, a TR can be viewed as a "laboratory", as a black box or as a subsystem, of object transformation.

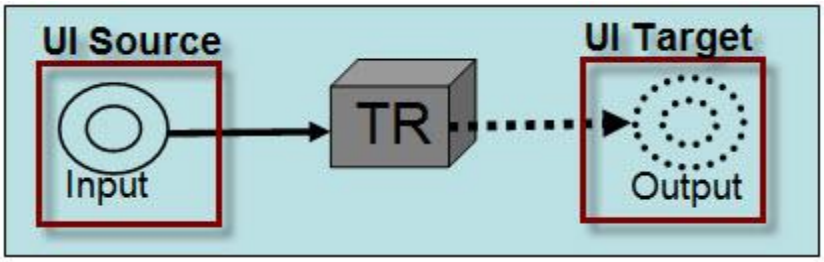


Figure 50. Transformation rule.

It can happen that an IO is associated to zero, one or many IOs. However, it should be insisted that the application of a TR provides a maximum of one IO. In the case of many choices, the first one is taken.

In each toolkit, a special object will be indicated. Under these conditions, it will be the object that will be chosen by default if the application of a TR finds no element. Therefore, a TR becomes a one-to-one relation.

Defining a *Transformation Atom (TA)* as a sextuplet <*Env,AT,Tk,Oc,Oi,s*> where :

1. *Env*: the environment, the platform or the model referenced by the user interface conception and/or the development. Let us recall that, in final level, a platform is a kind of foundation on which application programs can be written, read, executed and/or used. In general, a platform is composed of material, an operating system and software tools. At abstract and/or concrete level, a platform consists of models, tools and operators with which it can be possible to act on objects and/or transform them;

2. *AT*: the Application Type. We will see in the continuation that the type of application can influence the transformation of interactive objects. Using a

Personal Computer (PC) under Windows operating system, we count at least four types : Executable (Exe), Within Component (ActiveX DLL), Executable Component (ActiveX Exe) and Graphical Component (Ocx);

3. *Tk*: the Toolkit used to build the interface. A toolkit is a set of interactive objects. According to its objects, a toolkit can be abstract, concrete or final;

4. *Oc*: the concerned interactive object class. It is important to know the set of attributes which define the state of an interactive object before transforming it. It is also important to analyse methods which act on its behaviour;

5. *Oi*: the concerned interactive object instance. Each object is single. Two objects of the same class can lead to two different transformations. For example, in visual BASIC, the age and the name of the customer can be visualized by Text Boxes. A transformation can change the name into Text object and the age into Updown object.

6. *S*: is the set of dialogue scripts within Oi. To complete a transformation. It is necessary, to transport the object behaviour via its properties and scripts.

Thus, *Transformation Rule (TR)* can be seen as a kind of Many-to-Many relationship between Transformation Atoms in which semantic and functional values are the same both for the source and the target Atom i.e.

Transformation Rule Definition

TA=<*Env,AT,Tk,Oc,Oi,s*> → *TA'*=<*Env',AT',Tk',Oc',Oi',s'*>*
Where :

$$\begin{cases} \varsigma(\text{TA}) \cong \varsigma\,(\text{TA'}), & \varsigma \textit{ is the semantic function } (i) \\ £(TA) \cong £\,(TA'), & £ \textit{ is the syntactic function } (ii) \end{cases}$$

Finally, A *Transformation Mapping* (*TM*) is a set of Couples of Transformation Atoms

Mapping Definition

$\{(TA_i,TA_j)$: TA_j the image of TA_i i.e $TA_i \rightarrow TA_j\}$

Equations (i) and (ii) ensure mappings consistency. Indeed, the pseudo equation (i) guarantees the stability of the machine functional. the same input must produce the same results as in the source or target environment.

In addition, the pseudo equation (ii) requires a expressiveness continuity, also known as the syntactic consistency. The target environment should have at least the same expressive power as the source environment. Whether for the objects, the object properties or scripts, no information should be lost during mapping processing.

We insist on the fact that these two pseudo equations remain valid regardless of the type of mapping. For lateral mapping, models of the various views need to be syntactically and Semantically consistent with Each Other (*horizontal consistency*).

Similarly, for a forward or reverse mapping, a model must be consistent Semantically respectively with its specialization or its generalization (*vertical consistency*).

Respect of equations (i) and (ii) also guarantees the completeness of mappings. A complete specification in a source environment should be complete in the target environment. Thus, the following theorem is justified.

Completeness Theorem

If *TA*=<*Env,AT,Tk,Oc,Oi,s*> → *TA'*=<*Env',AT',Tk',Oc',Oi',s'*>* and *TA* is complete then *TA'* is complete.

Demonstration

Suppose that *TA'* is the result of a mapping application on *TA*. And, assume that *TA* is complete but *TA*' is incomplete. In this case, there would be :

- *An ontological problem*: there would exist statements whose meanings are inappropriate or absent in the target domain. This would be a contraduction of equation (i) which confirms that TA and TA' have similar semantic functions.
- *A linguistic problem*: there would be indescribable expressions in the target environment. In other words, the target language is poor compared to the source language. This would be a contradiction with equation (ii) confirming that the two languages have the same expressive power.
- *A Modelling problem*: the source would be more detailed than the target or vice versa. Therefore, equations (i) and (ii) would be violated, a contradiction of mapping definition.

In the Behaviour Editor, mappings are implemented very simply as shown in Figure 51 below. Indeed, the global properties of a mapping are: its *name*, its departure and arrival *levels*, its origin and arrival *toolkits*, the "*cardinal*" i.e. the maximum number of potential target objects associated with an object and its *ratio* which describes the numerical ratio, the scale, between source and target measurement units.

In the case of processing multiple situations where an object has two or more potential targets, regular expressions are used to avoid ambiguities or blockages.

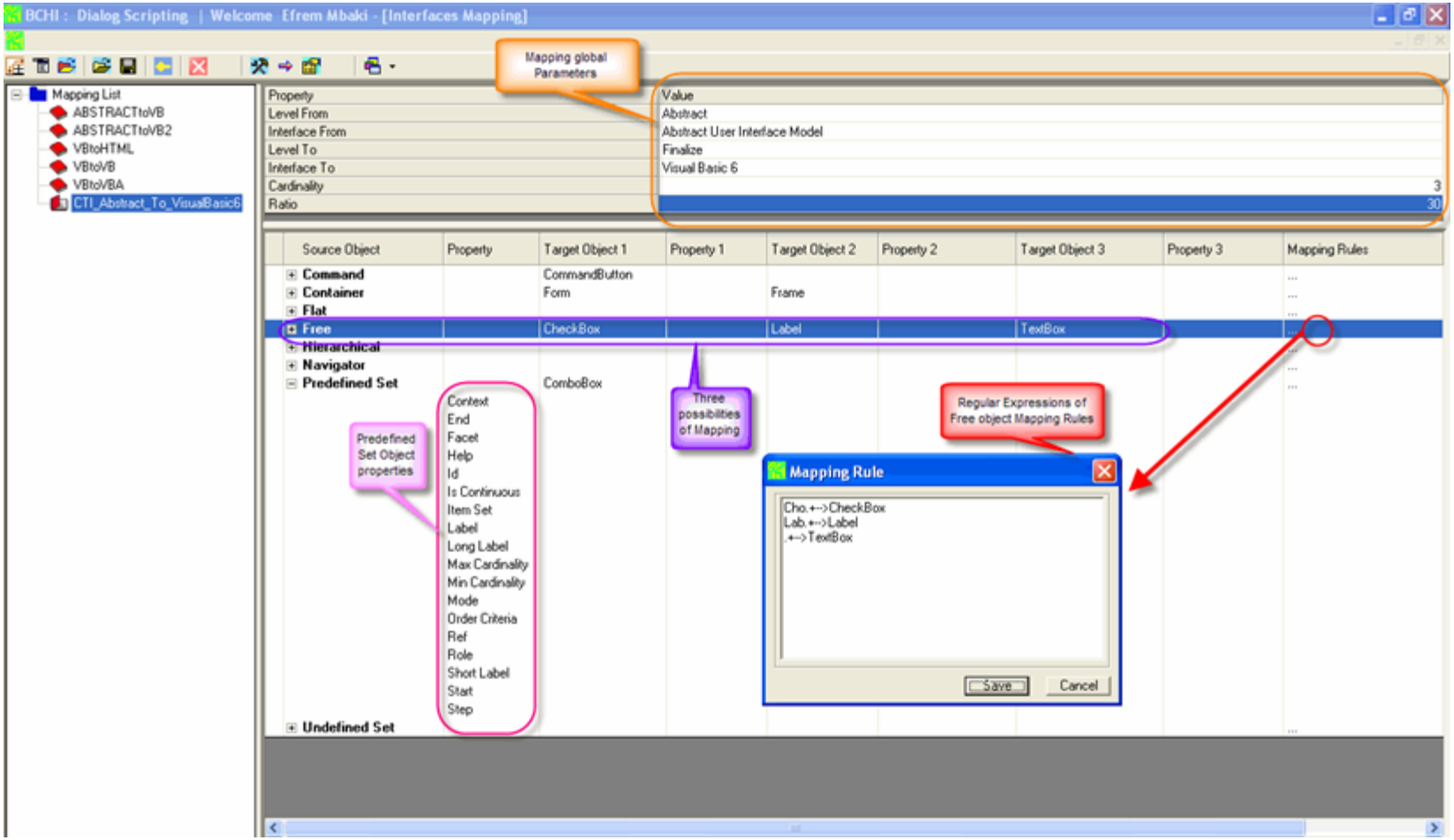


Figure 51. Mapping interface.

3.4 Conclusion

In this chapter, we have described a model-driven engineering of behaviours, essentially, in graphical user interfaces that is structured according to the levels of abstraction of the Cameleon Reference Framework. In fact, a dialogue model captures the abstractions of the behaviour as opposed to a traditional presentation model that captures the abstraction of the visual components of a user interface. The dialogue modelled at the abstract user interface level can be reified to the concrete user interface level by model-to-model transformation that can in turn lead to code by model-to-code generation. Three target markup and programming languages are supported: HTML, HTML Applications (HTA) and Microsoft Visual Basic for Applications (VBA). Two computing platforms are addressed: Microsoft Windows and Mac OS X. In this way, the approach demonstrates the capabilities of the abstractions in order to cover multiple programming paradigms and computing platforms.

Five levels of behaviour granularity are considered: object-level (behaviour of a particular widget), low-level container (behaviour of any group box), intermediary-level container (behaviour at any non-terminal level of decomposition such as a dialogue box or a web page), intra-application level (application behaviour) and inter-application level (behaviour across different interactive applications). Intra-container and inter-container behaviours are exemplified throughout a step-by-step methodology that is supported by a Dialog Editor, a model transformer and a code generator integrated into one single authoring environment. By translating into UsiXML (USer Interface eXtensible Markup Language) dialogue scripts built in this authoring environment, we obtain an effective solution of describing user interfaces and its behaviour with various levels of detail and abstraction and without limit about device, platform, modality and context.

Chapter 4 Applications of software support

For the validation, we use the *Dialog Editor* to develop software intended to cover the activities of Congo Transfer International (CTI) Company which is specialized in the international transfer of money and import express worldwide services. However, before addressing this complex example in the third subsection, the second section gives a complete presentation of tasks.

However, to better understand these two examples, we dedicate the first section to describing some basic concepts such as used by the script editor built into the Dialog Editor.

4.1 Basic samples

Here, we take the opportunity to illustrate basic examples related to five dialogue granularities described above. Let us specify the dialogues of a small application that evaluates a password. Appendix A provides a detailed description of this example. It manages interface and behaviour both in manual and automatic with the Dialog Editor.

4.1.1 Statement

Let us imagine a new user wants to connect to a system. For this, he/her must provide his name and also propose a login and password. Assume that the system provides a utility for the password evaluation. We confirm that this small example highlights the five granularities mentioned above.

Intuitively, the graphical interface of such a task would require three containers. In the first container, we will find input fields for the name, the login and the password. This container must also include three command objects which will be used respectively to save information, to solicit password evaluation or to exit the system. In the second container, we expect to find an input field, a graduate component that would indicate the security level of the password, an evaluation function of the password, two command objects to accept the changed password or to cancel the process. The third container is a query box which is activated when the user decides to leave the system. He is obliged to confirm before exiting. The five granularities are illustrated in this example as shown in the Figure 52 below.

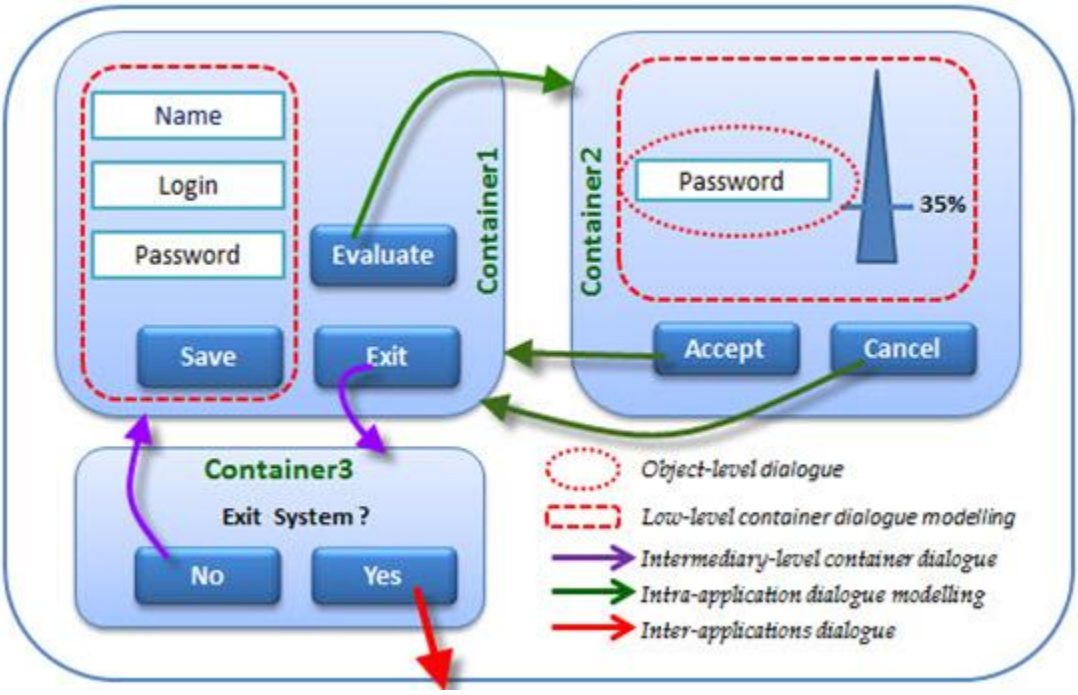


Figure 52. Dialogue granularity.

4.1.2 Dialogue granularity

1. *Object-level dialogue*: Each of the objects in these three containers has its own behaviour and therefore its dialogue script. We do not put the legend everywhere to avoid overloading the figure 52. Look in particularly at the password field in the second container. Indeed, any character addition or key change triggers the evaluation of the current password.

2. *Low-level container dialogue*: In the second container, the password behaviour influences the dynamic of graduated object. Indeed, the cursor is placed correctly in the graduated object according to the result of the evaluation of the password. Moreover, in a first container, the command object *save* is inactive as the name, the login or password is not filled in.

3. *Intermediary-level container dialogue*: If by error, or deliberately, the user chooses to leave container 1, a confirmation question is asked using the third container. A negative response will result in the normal pursuit of dialogue by *container1*.

4. *Intra-application dialogue*: if the user accepts the evaluated password, it becomes the current password. Otherwise, the password before the assessment remains valid.

5. *Inter-applications dialogue*: During third container interaction, if the user answers *yes*, he closes the system and goes to another task

4.2 Connection Sample

It should be recalled that the task is to enter a login and password. Two versions, in VB6 and HTML, of the expected and generated interface are given in Figure 53.

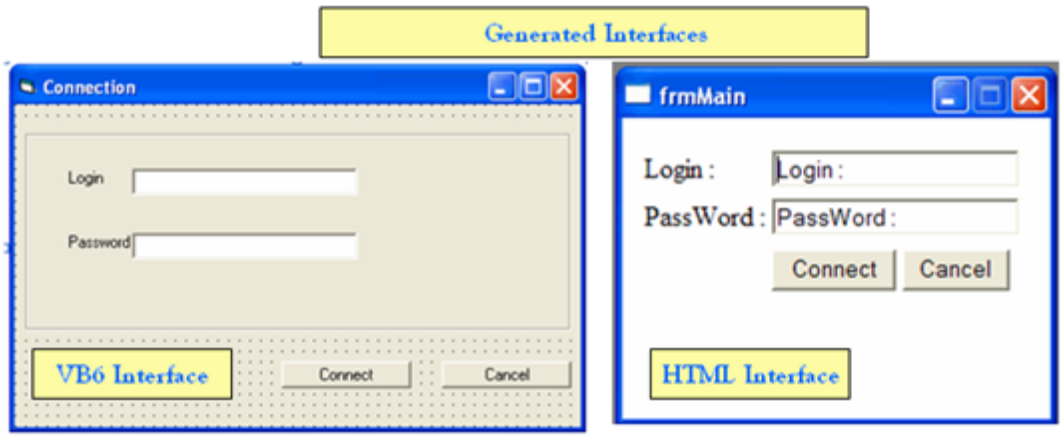


Figure 53. Final User Interfaces of Login & Password.

4.2.1 Project editing

Table 3 explains the main first steps for creating a new UI Project in the Dialog Editor, which basically consists of choosing the starting level of abstraction (typically, the AUI), the ending level (typically, one FUI), the toolkit, possibly with some extension and the library of mappings to be used. Note that one can start also at any other level such as FUI or CUI since multiple types of mappings are supported. For the login&password example, we limit ourselves to using five properties: two properties (i.e. left and top) determine the location of each interactive object, two other properties (i.e. height and width) specify the dimensions of each interactive object, and a fifth property (i.e. label) gives the object label text. The values of these attributes are taken into account during future transformations. Therefore, the resulting UI project holds the login&password with a quintuplet <*Label, Left, Top, Height, Width*> for each interactive object (Figure 53).

Table 3. Interactive objects of the login & password example.

IO Name	Parent	Description	Type	Properties
frmExist		Main Form	Contener	<3615,60,450,6360,Connection>
fraIdent	frmExist	Secondary Form	Container	<2295,120,240,6135>
lbLogin	fraIdent	Login invitation	Free	<300, 480,480,1000,Login>
txtLogin	fraIdent	Login contain	Free	<300, 1200,480,2535>
lbPwd	fraIdent	Password invitation	Free	<300, 480,1200,1000,Password>

txtPwd	fraIdent	Password contain	Free	<300, 1200,1200,2535>
btnConnect	fraIdent	Validation Trigger	Command	<300, 3000,2880,1455,Connect>
btnCancel	frmExist	Cancel Trigger	Command	<300,4800,2880,1455,Cancel>

4.2.2 Project transforming

Let us assume that we want to apply Model-to-Model transformation (M2M) from AUI to CUI. For this purpose, Figure 53 lists some mappings that have been implemented for this purpose, here for a vocal UI and a GUI, both appearing at the CUI level: Container is translated to questionnaire/Form if its name begins with frm or SubQuestionnaire/SubForm if its name begins with fra. Free object change to Request/Label if its name begins with lb or to Answer/Text Box if its name begins by txt.

The command object is expressed as verbal validation or a button depending on the interaction modality. By applying the mappings for a GUI, we obtain a CUI with a graphical modality.

4.2.3 Code generating

In order to transform this CUI into a FUI (say here that we want both the VB6 and HTA GUIs), Table 4 and Table 5 list some mappings that have been implemented.

Table 4. Mapping from Abstract to Concrete

Abstract UI	Vocal UI	Graphical UI
Container	Frm* → Vocal Quiz Fra* → Sub Vocal Quiz	Frm* → Form Fra* → Sub Form
Free	Lb* → Request Txt* → Answer	Lb* → Label Txt* → Text
Command	Validation	Button

Table 5. Mappings from Concrete to Final User Interface.

Graphical UI	Visual Basic UI	HTA UI
Form / Sub Form	Frm* → Form Fra* → Frame	Frm* → Form/Page Fra* → FieldSet
Label / Text	Lb* → Label Txt* → Textbox	Lb* → Input (Text) Txt* → Input (text or password)
Button	CommandButton	Button
Ratio	1	0.05

4.2.4 Conclusion

This small example relative to system connection has been of great benefit. Indeed, it has allowed to map and to implement the various concepts that we present in this thesis. The values presented above were introduced in the editor and produced the expected results.

4.3 CTI Application

With *Dialog Editor*, we developed software intended to cover the activities of a company which is specialized in the international transfer of money and import express worldwide services. We needed an application based on a real case. But, we also needed an End-User oriented dialogue application. For these two reasons, we asked the employers of CTI Company, whose head office is located in Liege in Belgium, to participate in the analysis and the validation of this software. Insofar as the employees of CTI Company speak only French, interfaces are in French in order to allow better communication. In this context, commercial transaction is defined by: a Shipper (the customer which deposits the money or the object), a Sender (the person for whom the money or the object is intended) and An Order (the details of the transaction contents).

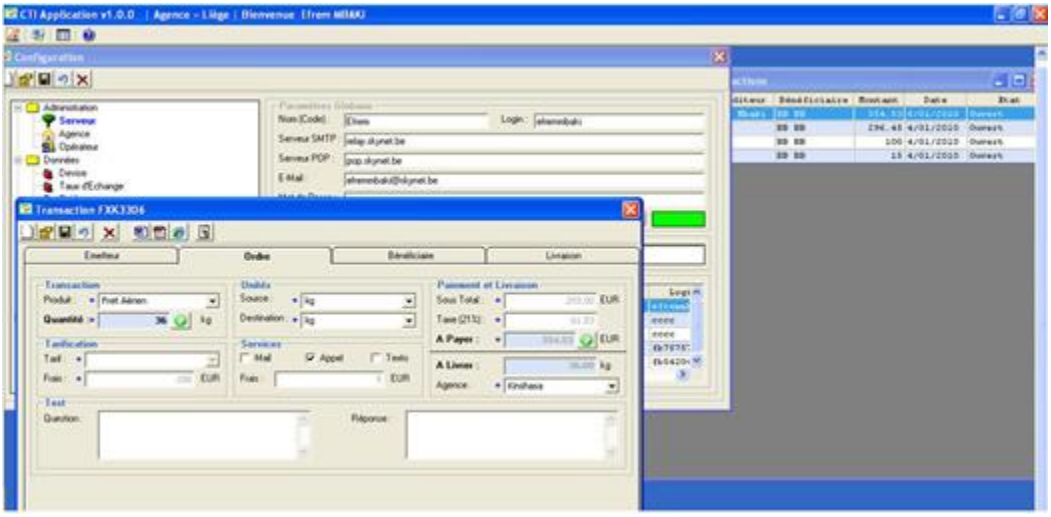


Figure 54. Global view of CTI Application.

4.3.1 Software components

The objective of this subsection is to present the software components of the CTI application. Indeed, the architecture has four primary component parts, as shown in the Figures 55, 56 and 57 : Administration, Data, Transaction and Reporting.

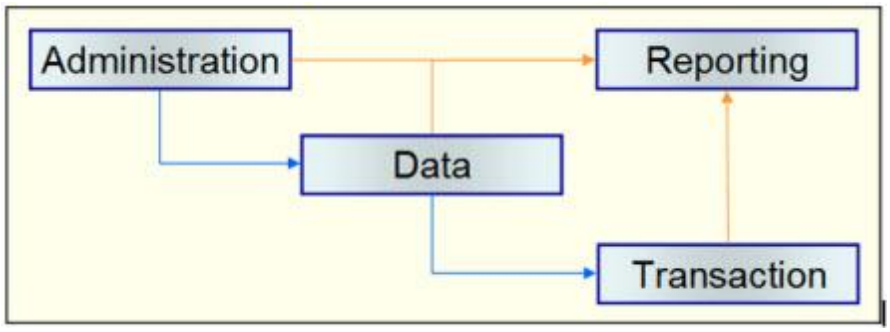


Figure 55. CTI application components.

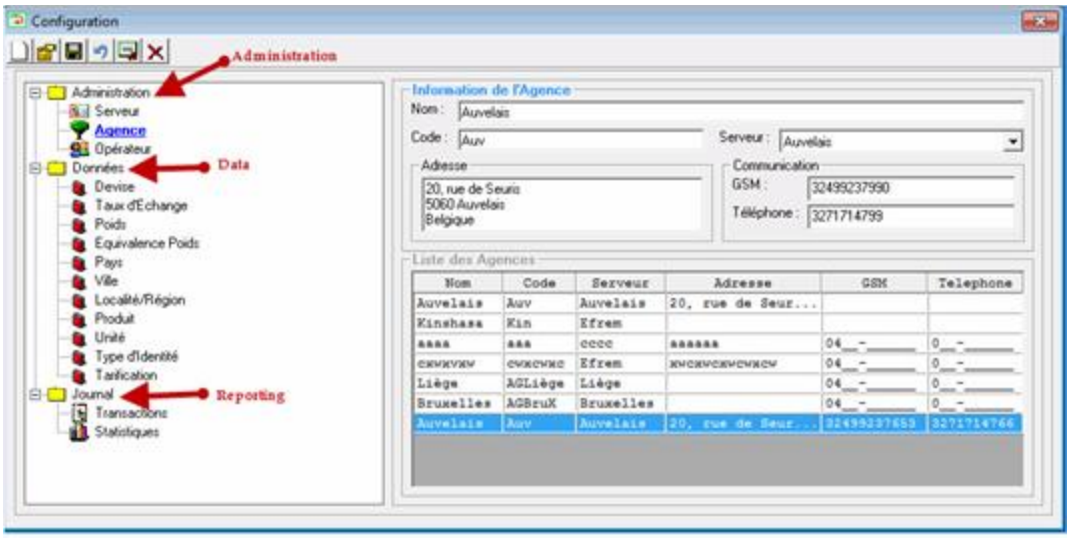


Figure 56. CTI Configuration UI.

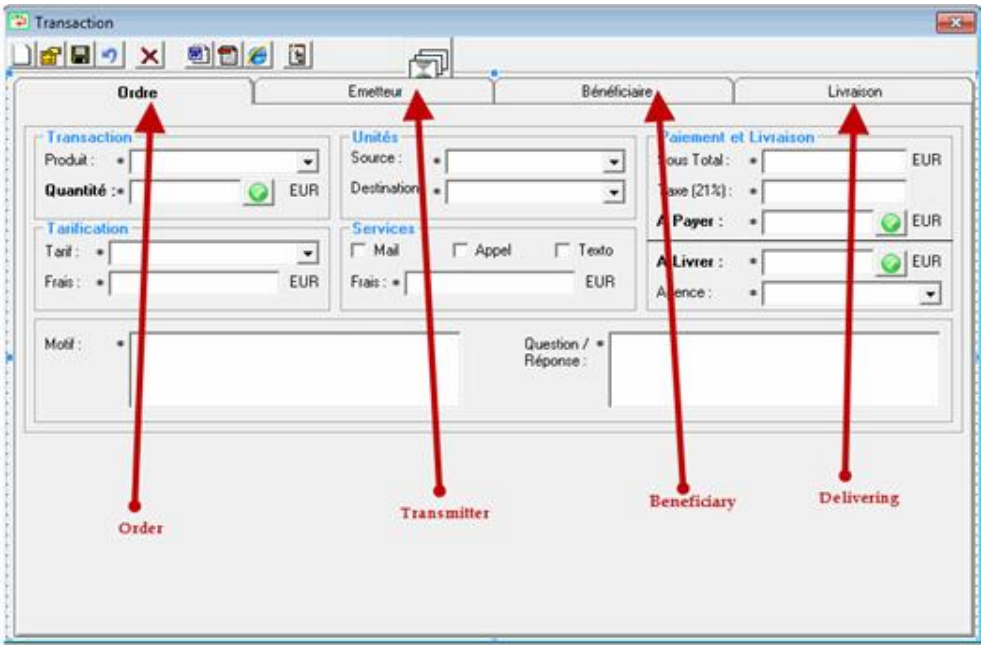


Figure 57. CTI Transaction UI.

The entrance point is the Administration Module. It should be pointed out that mailing is the communication chosen for information exchanges in this application. As shown in the Figure 58, each CTI agency is identified by an address email. The Administration module offers the functionalities for encoding and checking the correction of SMTP and POP server addresses.

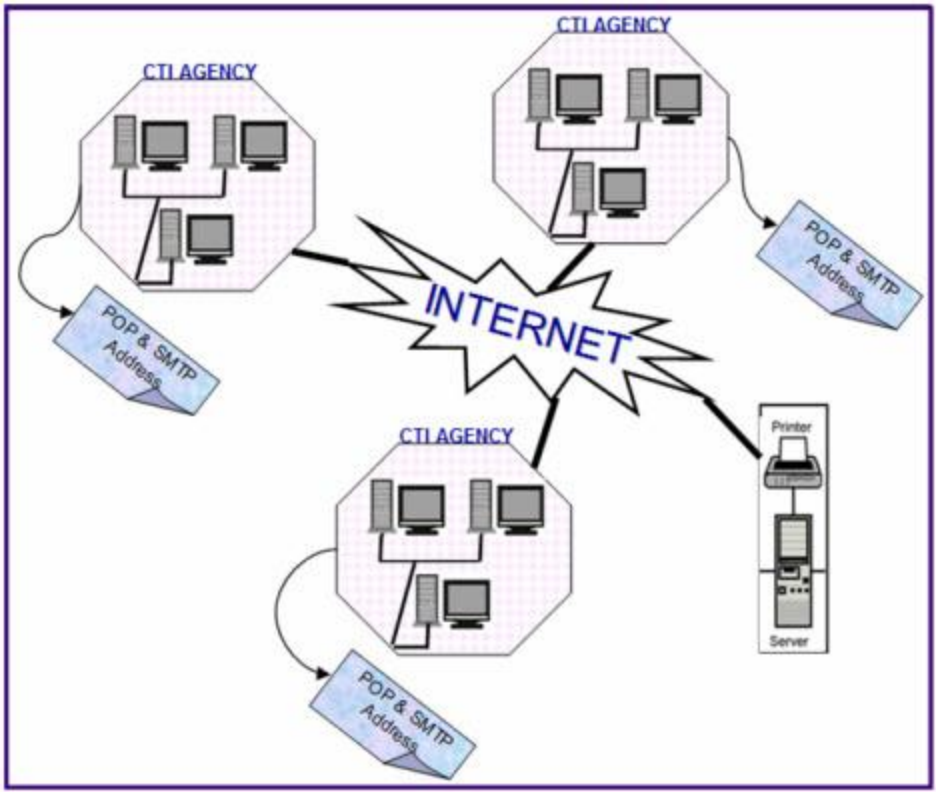


Figure 58. CTI network agencies.

Only the administrators have the right to create/modify these properties. Once saved, inserted data and/or modifications are disseminated to all the agencies. In addition to the technical aspects, the administrators can also create/modify operators and attribute rights to each agency. This module is dependent on Administration module for the simple reason that only administrators have the right to add or modify these data.

The Data module offers the functionalities to manage benchmark data such as currencies, countries, product types, cities, weights or types of identity cards. However, it also makes it possible to recover market data such as exchange rates or weight equivalences.

The Transaction module is controlled by the operators. It proposes functions to manage financial transactions. A transaction requires four pieces of information shown in the Figure 58:

1. *Order*: which product? How much to deliver? And where?
2. *Transmitter*: who orders? Who buys?
3. *Beneficiary*: who receives the order?
4. *Delivering*: who, when and where for the delivering

The CTI application has a powerful Reporting module. Indeed, according to the different scenarios in use, this software offers the operators the possibility to export some data to Ms Word, Html or Pdf documents.

Moreover, administrators are able to make remote requests, in real time, for the current daily or periodical financial statement of any agency.

4.3.2 Transaction Order data structure

To record a transaction order, an operator needs the following information: Firstly, he must fix the type of product desired by the Customer. Possible values are Express services, Air freight, Ocean freight or Money Transfer. Secondly, the CTI Operator must ask to customer to fix the quantity or amount concerned. According to the type of product and the quantity, the system determines the applicable tariff and thus, the reference unit (currency) and the tariff price of service. If the delivery unit (currency) is different from the reference unit, the system provides the conversion rate automatically and adapts the tariff price upon request, prices of services of follow-up (email, text-message, telephone call), intermediate total, the tax to be paid (21%) and then the total price.

To finish the transaction, the operator must ask the customer for the town of delivery and, if required, the question to be asked during the delivery and the corresponding answer. We can summarize this information via the following UML diagram UML, Figure 59.

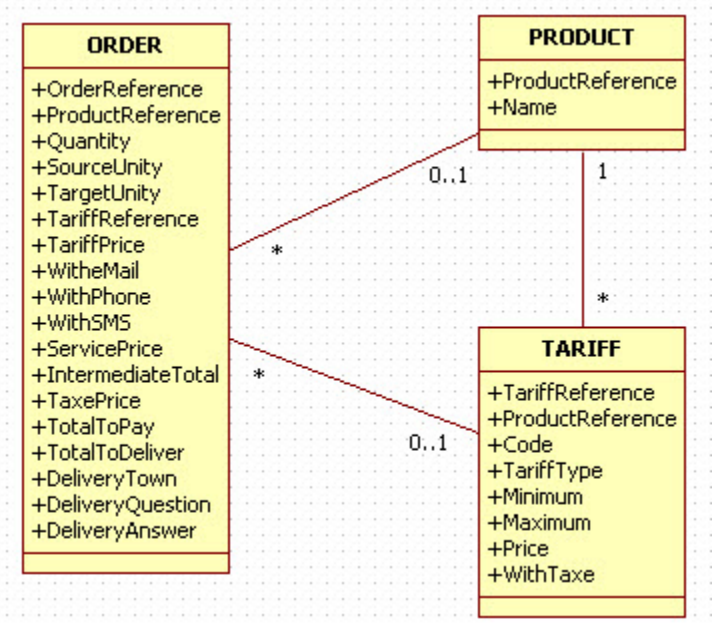


Figure 59. CTI Order by UML data model.

4.3.3 Using Dialog Editor

By implementing the software "CTI Application", we had to use the *Dialog Editor* to generate part of the code by the means of specification. To get an idea of what we have

lost or gained by choosing to work with the editor dialogs, we set out to observe two parameters, the time spent and the number of lines in the code.

In fact, we have worked 181.5 man-days to implement CTI software by exploiting partially the Dialog Editor. The first table below gives a temporary outline of the tasks carried out and, expressed as a percentage, indicating the parts carried out manually and those generated automatically.

Table 6. Tasks time distribution.

Task	Timing	Manuel	Automatic
Interactive Objects library implementation	25%	100%	**0%**
Shipper, Sender and Order Management	65%	20%	**80%**
Reporting Management	**10%**	**50%**	**50%**

If I had to work manually, my long experience as a developer would help to finish the implementation of this software for, approximately, 181.5 main-days. Clearly, there is a huge waste of time. The rigor of Dialog Editor requires a significant effort in the specification of each item in the presentation and/or the dialog module. Although expected, this loss can be relativized if the software could be re-written from one environment to another. Moreover, the gain is real concerning the number of code lines. Dialog Editor offers facilities in specifying in "two clicks", possibly, with a few number of words a large number of instructions. The best quality of the Editor Dialog interpreter is to be be able understand short sentences to generate the appropriate code. The following two tables provide details on what we have received respectively about the the number of code lines and spent time.

Table 7. Spent time for CTI Application.

File				SPENT TIME		
				Manual Programming	Automatic Programming	
Type		Number			Specified part	Programmed part
License		3		1	1.5	0
Database		1		5	0	0
User Interface (UI)		40		40*2=80	40*2.5=100	
Dialogue Script	Business object	14	56	56*1.5=84	56*1=56	56*0.5=28
	UI Control	40				
	Data access	1				
	Technical	1				
Project		5		5*0.2=1	5*1=5	0
User control		10		10*1=10	0	10*2=20
Total		**115 Files**		**181 man-days**	**162.5 man-days**	**48 man-days**
Percentage				**100%**	**90%**	**26%**
Loss				**0%**	**-16%**	

Table 8. Code lines number for CTI Application.

File				CODE LINES		
				Manual Programming	Automatic Programming	
Type		Number			Specified part	Programmed part
License		3		35	3	0
Database		1		0	0	0
User Interface (UI)		40		40*50=2000	40*40=160	
Dialogue Script	Business object	14	56	56*60=3360	56*30=1680	56*20=1120
	UI Control	40				
	Data access	1				
	Technical	1				
Project		5		5*5=25	5*4=20	0
User control		10		10*70=700	10*30=300	10*100=1000
Total		**115 Files**		**6120 Lines**	**2163 Lines**	**2120 Lines**
Percentage				**100%**	**35%**	**35%**
Benefit				**0%**	**30%**	

Table 2 is more explicit. It establishes clearly the statistical elements by comparing the number of files and the numbers of VB code lines of the by working manually and by using Dialog Editor. Table 2 provides clear evidence of a 30% benefit when number of files and code lines are compared.

4.4 Conclusion

The example of connection explained at the beginning of the chapter has highlighted five dialogue granularities. In continuation of this example, we gave important information about the operation of the Dialog Editor for specifying the exchange.

The subsection relative to CTI Application illustrated some important concepts in the specification and design of an interactive task such as software architecture and the dialog automata. Moreover, we exhibited some advantages and disadvantages when working manually or getting help from the Dialog Editor.

If current chapter was interested in aspects of performance, the next chapter will analyse the ergonomic aspects of the *Dialog Editor*.

Chapter 5 Quality characteristics of Dialog Editor

The main objective of this fifth chapter is consideration of the usefulness and usability of *Dialog Editor*. In effect, the third chapter uses the UML language to describe in detail the conceptual model and defines all its objects. In the same way, it provides the flux diagram of the methodology, while at the same time specifying the consistency and completeness properties of the transformation model. The last section of this major chapter is devoted to the description of *Dialog Editor*, the software that we have used in the framework of our research.

Furthermore, the fourth chapter illustrates the use of *Dialog Editor* through a simple example (connection to an interactive system with a login and a password) and another more complex example (*CTI Application,* relative to CTI Company*,* intended to cover the activities of a company which is specialized in the international transfer of money and import express worldwide services). These two examples of use have revealed strong points of *Dialog Editor*, as well as certain gaps.

To pursue this approach, the current chapter focuses on the evaluation of the results proposed in the third chapter. To that end, we have opted for a qualitative approach. Our assessment plan is structured in three phases, each of which is the subject of a section of this chapter.

In the first instance, in the form of an interview we presented an open questionnaire to the potential users, in order to have feedback of a more or less general nature. Then, we organised a satisfaction survey by making use of the *IBM Computer Usability Satisfaction Questionnaires (CSUQ)* [Lew95], a closed and fairly structured questionnaire. It is worth recalling that the CUSQ proposes a validated empirical approach. It has a correlation factor of 0.89 as far as usability of an interface is concerned. Moreover, it aggregates four metrics, the study of which will allow the extrapolation of certain explanations as to the utility and usability of *Dialog Editor*. Finally, to conclude this assessment, we have applied the respondents' comments to the ISO / CEI 9126 norm criteria to have a clearer vision of the useful and usable characteristics of Dialog Editor.

5.1 The Interviews

With the help of a short and entirely open questionnaire, the idea has been to observe the behaviour of a fairly representative group of users and to draw conclusions based solely on those observations.
More precisely, we sought to measure user satisfaction as concerns the use of Dialog Editor, so as to reveal some of its strengths and weaknesses. However, we should emphasise the fact that *Dialog Editor* is not a commercial product but a prototype created to show the feasibility and methodology that we are proposing.
Depending on the setting for the interview, we were able to allow the respondent to use the software and then ask questions or first explain how *Dialog Editor* works and then proceed to a question-answer session. However, in order to set the framework for the discussion in the best possible way, each interview began by a brief explanation of the methodology. We then proceeded, in an entirely random way, to use one of the three types of interview:

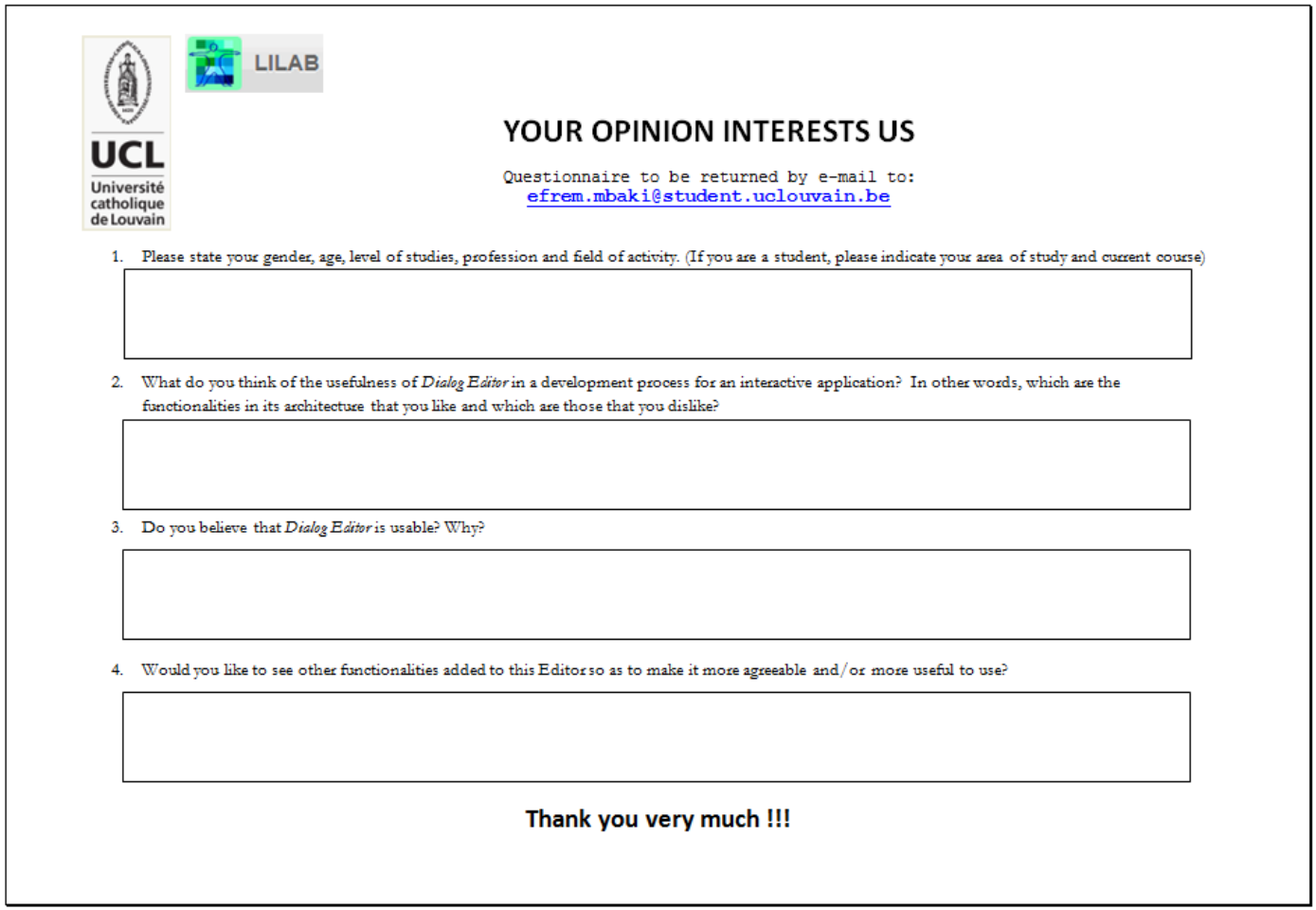
1. *Free*: the respondent comments freely on the tool, without any need to be asked questions ;
2. *Guided*: a list of questions is put to the respondent who replies once he/she has used the software, and
3. *Semi-structured*: a mix of free and guided interview.

Furthermore, it is important to note that we interviewed some people in groups and some individually. In the first instance, we turned to our professional colleagues and those in the laboratory. We interviewed former university friends from the University of Namur or other people presented to us. Altogether, we had around 20 respondents.

The rest of this section is organised into four parts, the first of which presents the questionnaire used, the second describes the demographic data of the respondents, the third analyses the replies given by those taking part and the fourth concludes the experiment.

5.1.1 The questionnaire

As it was never our intention to formalise a subjective assessment, we limited ourselves to preparing a questionnaire which was both simple and short, so as to reduce to a minimum the time needed to answer, thereby privileging conviviality during the discussion. Thus, the basic questionnaire contained four questions concerning respectively respondent data, his/her opinion as to the usefulness of the tool, her/his opinion as to its usability and the list of possible extensions he/she proposed for Dialog Editor.



UCL Université catholique de Louvain

LILAB

YOUR OPINION INTERESTS US

Questionnaire to be returned by e-mail to:
efrem.mbaki@student.uclouvain.be

1. Please state your gender, age, level of studies, profession and field of activity. (If you are a student, please indicate your area of study and current course)

2. What do you think of the usefulness of *Dialog Editor* in a development process for an interactive application? In other words, which are the functionalities in its architecture that you like and which are those that you dislike?

3. Do you believe that *Dialog Editor* is usable? Why?

4. Would you like to see other functionalities added to this Editor so as to make it more agreeable and/or more useful to use?

Thank you very much !!!

Figure 60. Open questionnaire used for interviews

5.1.2 Demographic data

During these interviews, we had the privilege to solicit personal information from respondents to classify their profiles. The objective of this section is to present the various graphs showing respondents according to gender, age, field of activity, profession or level of studies.

	Frequency	Percent	Valid percent	Cumulative percent
Man	8	40,0	40,0	40,0
Woman	12	60,0	60,0	100,0
Total	20	100,0	100,0	

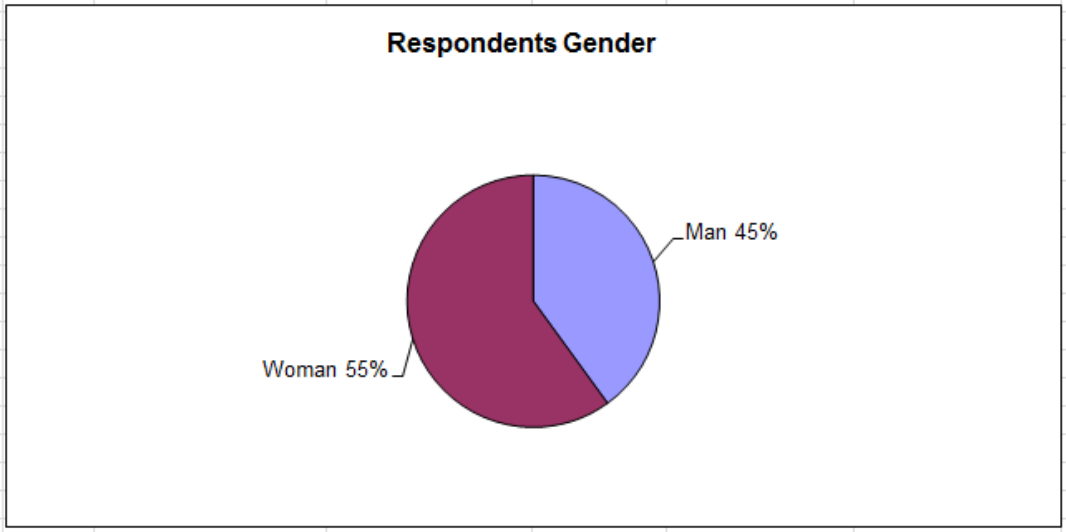


Figure 61. Respondents Gender

Figure 61 shows that 55% of respondents were women and 45% of them were men. It is also established that the average age of respondents was 34.35 years old (Figure 62).

According to Figure 63, despite our efforts, 80% of interviewees have university degree. This is understandable insofar use the Dialog Editor requires a significant background in programming or in the use of interactive applications.

Referring to Figure 64 which describes the occupations of respondents, we note that the majority of them worked in the financial field (Employee 40% and Executive 15%) or are researchers (Students 30%). Among them, we have to meet people with a long practical experience. They were so practical that it was difficult to explain theoretical concepts such as the abstract interface. We were also pleasantly surprised by the importance of a good explanation. We were pleasantly surprised by the intuition of some respondents. In fact, by intuition and by the structure of *Dialog Editor* interface, they understood beyond our explanation.

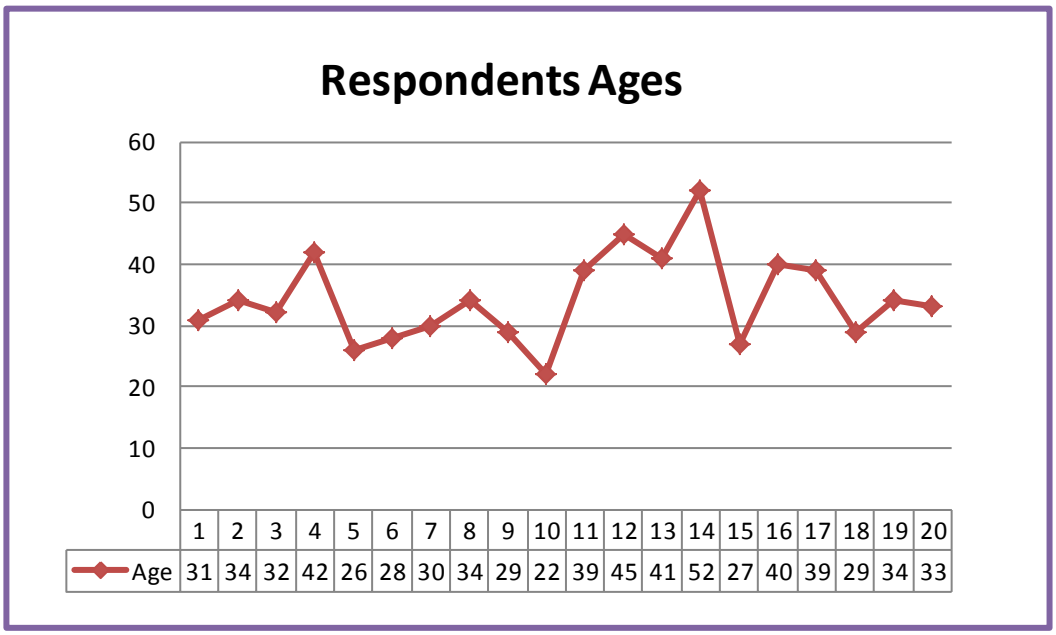


Figure 62. Respondents Ages

	Frequency	Percent	Valid percent	Cumulative percent
Primary	0	0,0	0,0	0,0
General secondary	1	5,0	5,0	5,0
Technical secondary	0	0,0	0,0	5,0
University graduate	16	80,0	80,0	85,0
Post-graduate	2	10,0	10,0	95,0
Professional	0	0,0	0,0	95,0
Other	1	5,0	5,0	100,0
Total	20	100,0	100,0	

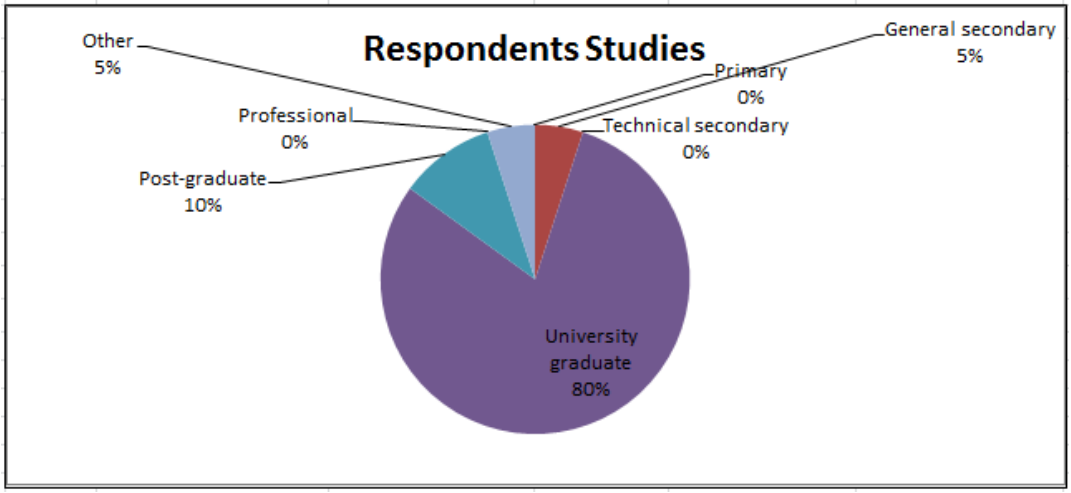


Figure 63. Respondents Studies

	Frequency	Percent	Valid percent	Cumulative percent
Student	6	30,0	30,0	30,0
Executive	3	15,0	15,0	45,0
Self-employed	0	0,0	0,0	45,0
Financial	0	0,0	0,0	45,0
Manual worker	1	5,0	5,0	50,0
Civil servant	1	5,0	5,0	55,0
Employee	8	40,0	40,0	95,0
Retired	0	0,0	0,0	95,0
Other	1	5,0	5,0	100,0
Total	20	100,0	100,0	

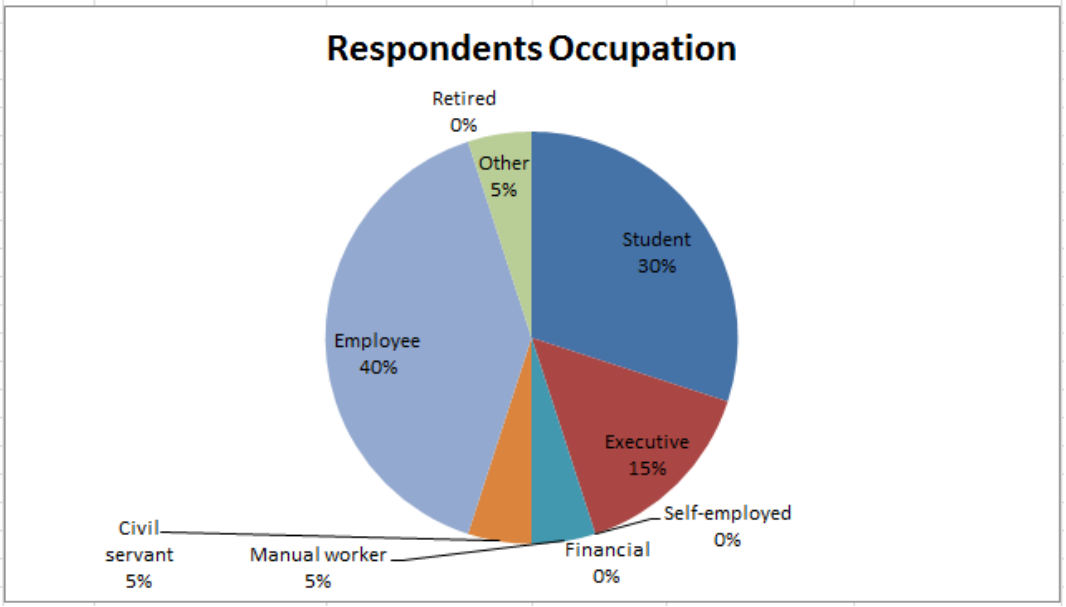


Figure 64. Respondents Occupation

5.1.3 Analysis of replies

The replies provided by those taking part were analysed according to the three main aspects which emerged during the interviews, i.e. the analysis and design of Dialog Editor, the models used in Dialog Editor and the code generation proposed by Dialog Editor.

1. Analysis and design

It is established that Dialog Editor is for a single user profile. Nevertheless, those people interviewed were not deterred by this fact. In fact, this weakness is lessened, on the one hand, by the proposed traceability management in the software and, on the other hand, by the possibility to work at abstract, concrete or final level. However, one respondent suggested the implementation of a system to block script when validated, so as to stop anyone else from changing it subsequently.

Furthermore over, four major steps in the methodology were clearly identified by the persons interviewed. They unanimously approved the clarity of the approach and the succession of models, even if certain expressed doubts about the script editor which, it should be said, has not been entirely finished.
To sum up, as the table below shows, twelve persons interviewed adopted the methodology elements, as well as *Dialog Editor*. One person approved the tool but expressed doubts as to the effectiveness of the proposed models. Four people declined to make any comment and three people approved the theoretical aspects but found the software to be inoperable for complex tasks. It should be noted that this last group of people had difficulty integrating the concept of an abstract interface.

Table 9. Analysis and Design Survey Feedback

Acceptance	60%
Doubt	5%
No comment	20%
Refusal	15%

2. The modelisation

We should point out here that we have integrated into the editor database, amongst other things, the abstract model proposed by the Moskitt Group. With this abstract model, the interviewees visualised the methodological approach, but, due to their training and long practical experience, they experienced difficulty imagining an abstract interface. Consequently, it was difficult for them to understand the concept of the abstract interface.
Therefore, in order to take full advantage of the comments and suggestions made by those questioned, we chose to centre discussion at the final level. In fact, the statistics show that fifteen people gave their approval to the modeling, one person disapproved and four people made no comment.

Table 10. Modeling Survey Feedbacks

Acceptance	75%
Disapproval	5%
Without comment	20%

3. Code generation

The majority of people that we interviewed are developers. As such, they were very interested in the code generation module. In fact, *Dialog Editor* generates code according to the choice of the user. In addition, the code generated respects the separation of three parts: the interface, the operating machine and the dialogue controller.
Those interviewed had the opportunity to point out the difficulty of saving modifications made manually. In effect, at the time of generation, everything is cleared before being automatically reconstructed. It is therefore impossible to quarantine codes manually, with a view to inserting them in the correct place at a later stage. Moreover, all persons inter-

viewed regretted the fact that the specification is principally textual, and particularly regretted this aspect when it came to the placement of interactive objects.
Eighty per cent of respondents approved the functionalities proposed by the code generator. Better still, they even noticed a reduction in the number of lines of code as compared to manual programming.

Table 11. Code Generation Survey Feedback

Acceptance	**80%**
Disapproval	20%

It should be said that four persons had serious doubts. They believed that this tool would not operate for complex applications.

4. Conclusion

Although the method used is empirical, the results of the inquiry do confirm the usefulness and the usability of *Dialog Editor*. Indeed, the models are intuitive and easy to use. However, a teaching effort is required to better explain the abstraction of interfaces. The methodology of *Dialog Editor* is integrated in a transparent way into the four modules. Its code generator increases productivity while at the same time reducing errors to a minimum. Certain functions have been proposed with a view to possible extensions. We will proceed to list them in the general conclusion. Some weaknesses as to the ease of use of *Dialog Editor* have also been revealed, particularly with regard to the placement of interactive objects.

5.2 Satisfaction survey

The satisfaction survey has an important place in quality management. Taking account of client satisfaction is one of the major preoccupations today for most businesses. It is at the heart of the new ISO 9000 (2000) norms. For Human-Machine interfaces, we talk about usability tests which consist of an empirical assessment based solely on the experience of users to « measure » effectiveness, learning rate, risk of error or many other variables.

The aim of the survey carried out is to measure user satisfaction concerning usefulness and usability. The different measures were obtained from replies to the *Computer Usability Satisfaction Questionnaires (CSUQ)* [Lew95] designed by IBM. We should insist on the fact that these replies to the CSUQ have been expressed using a Likert 7-point scale where 1 represents the worst perceived rating (strongly in disagreement), and 7 represents the best perceived rating (strongly in agreement) [Lik32].

The *CSUQ* is essentially made up of 19 questions, structured in four parts, as follows:

1. The first 8 questions deal with the usefulness of the system (*System Usefulness, SysUse*). The interviewee replies give an indication of the presence or absence of expected services, as experienced by the users ;

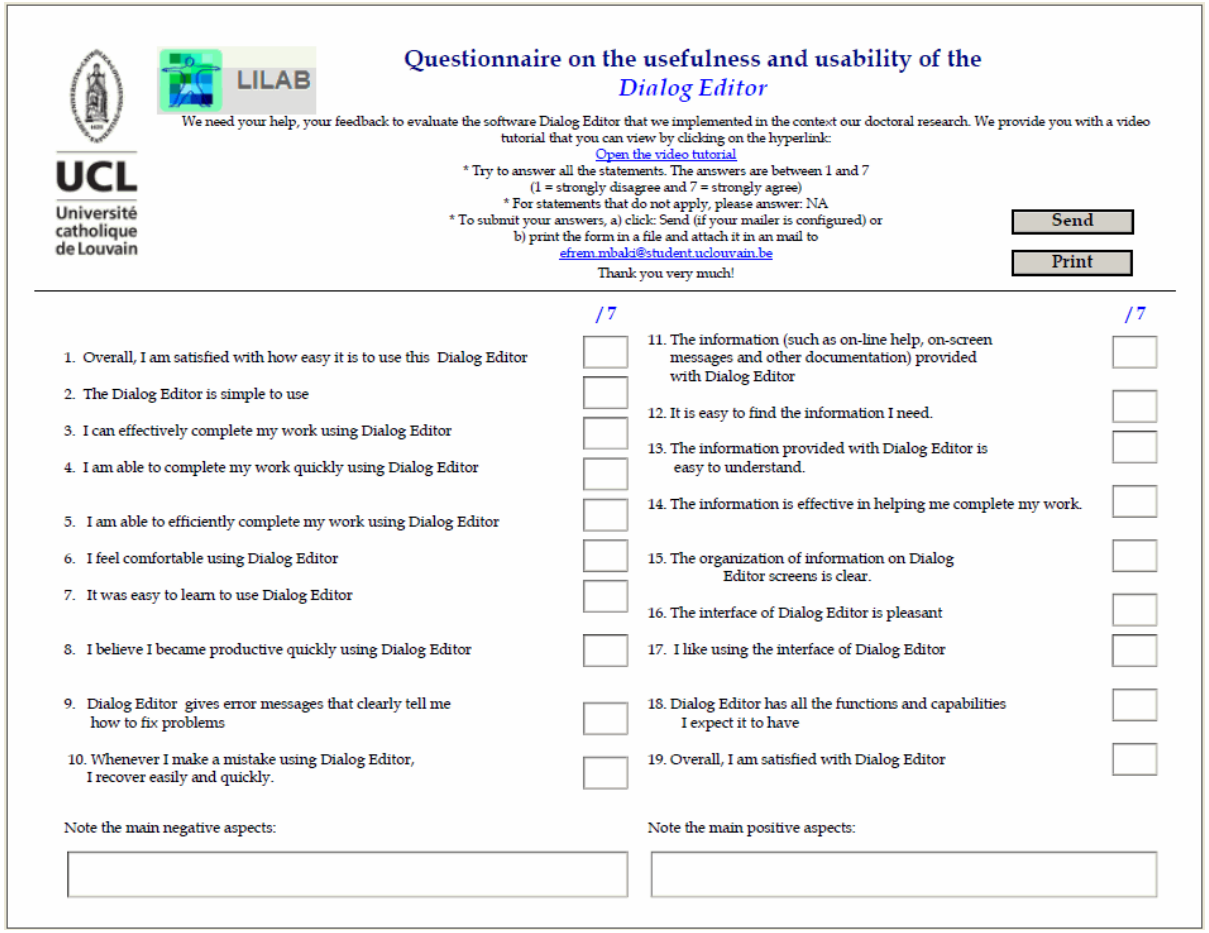
2. Questions 9 to 15 concern the quality of information (*Information Quality, InfoQual*). Here it is hoped to have an insight into the pertinence of the information proposed about the interfaces ;

3. Questions 16, 17 and 18 concern the quality of the interface (*Interface Quality, IntQual*). The interviewees' replies allow us to assess satisfaction as far as presentation of the interactive system under evaluation is concerned ;

4. The last question concerns the global view (*OVERALL*) which takes into account all indicators, so as to summarise overall interviewee satisfaction.

CSUQ also contains open fields to list at most three extremely positive aspects or three extremely negative aspects of the system under assessment. We extended the *CSUQ* questionnaire to other factual questions, such as gender, age, field of activity, profession or level of studies of the interviewees.

We organised the rest of this sub-section under three points : the first deals with interview modalities. The second presents the results obtained and launches a discussion on the elements of reply, and we finish with a conclusion.



UCL Université catholique de Louvain

LILAB

Questionnaire on the usefulness and usability of the *Dialog Editor*

We need your help, your feedback to evaluate the software Dialog Editor that we implemented in the context our doctoral research. We provide you with a video tutorial that you can view by clicking on the hyperlink:

Open the video tutorial

* Try to answer all the statements. The answers are between 1 and 7 (1 = strongly disagree and 7 = strongly agree)
* For statements that do not apply, please answer: NA
* To submit your answers, a) click: Send (if your mailer is configured) or b) print the form in a file and attach it in an mail to efrem.mbaki@student.uclouvain.be

Thank you very much!

Send

Print

	/7		/7
1. Overall, I am satisfied with how easy it is to use this Dialog Editor		11. The information (such as on-line help, on-screen messages and other documentation) provided with Dialog Editor	
2. The Dialog Editor is simple to use		12. It is easy to find the information I need.	
3. I can effectively complete my work using Dialog Editor		13. The information provided with Dialog Editor is easy to understand.	
4. I am able to complete my work quickly using Dialog Editor		14. The information is effective in helping me complete my work.	
5. I am able to efficiently complete my work using Dialog Editor		15. The organization of information on Dialog Editor screens is clear.	
6. I feel comfortable using Dialog Editor		16. The interface of Dialog Editor is pleasant	
7. It was easy to learn to use Dialog Editor		17. I like using the interface of Dialog Editor	
8. I believe I became productive quickly using Dialog Editor		18. Dialog Editor has all the functions and capabilities I expect it to have	
9. Dialog Editor gives error messages that clearly tell me how to fix problems		19. Overall, I am satisfied with Dialog Editor	
10. Whenever I make a mistake using Dialog Editor, I recover easily and quickly.			

Note the main negative aspects:

Note the main positive aspects:

Figure 65. CSUQ questionnaire used for the satisfaction survey

5.2.1 Methodology

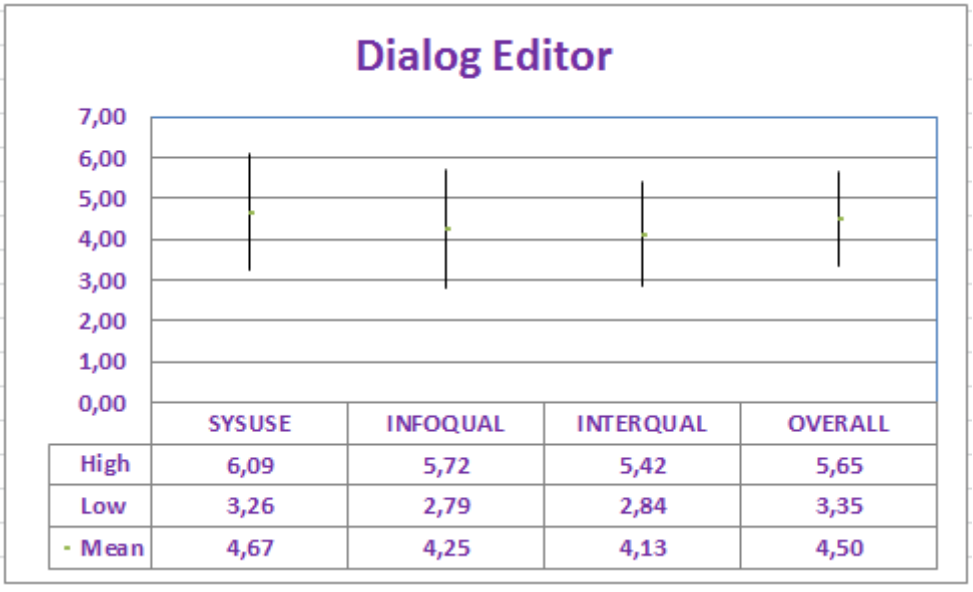
We drew up the *CSUQ* questionnaire with the logos of the Université Catholique de Louvain and the LiLab Laboratory before sending it to the interviewees by e-mail. An explanatory video was made available to them, able to be downloaded from the UCL server. As far as possible, we met with certain interviewees for a demonstration and/or a theoretical discussion. Three weeks later, around 20 duly-completed questions were returned to us by e-mail. We then proceeded to encode this data in an Excel file, from which we extrapolated the statistical formulae and tables in the section that follows.

We insist on the fact that negative and positive comments made by respondents are not very significant. In other words, no major point seems to attract the attention of respondents in the positive or in the negative. Nevertheless, we integrate these remarks in the argument of the next section regarding the application of the criteria of ISO/IEC 9126, specially in the usability evaluation.

5.2.2 Results and discussions

1. Global parameters

The table of data shows a relative dispersion among the users, whether it be for the usefulness of the functionalities, the quality of the information or even the quality of the interface. Thus, looking vertically at the table below, we can see quite a gap between the lowest score, the highest and the average or mean. On the other hand, looking at it horizontally, the values remain more or less stable, whatever the variable being measured. In short, by looking at the averages or means below, the overall perception of those interviewed is slightly higher than 4 out of 7.



	SYSUSE	INFOQUAL	INTERQUAL	OVERALL
High	6,09	5,72	5,42	5,65
Low	3,26	2,79	2,84	3,35
Mean	4,67	4,25	4,13	4,50

Figure 66. CSUQ Parameters for Dialog Editor

Out of four general parameters, *Sysuse* is the one obtaining the best score (6.09). However, there is a gap of 1.42 when compared to the average. This means that opinions are divided as far as the usefulness of *Dialog Editor* is concerned. Nevertheless, with an average of 4.67 (out of 7) we can conclude that the interviews approved the functionalities of Dialog Editor.
The *Interqual* parameter is that which obtains the least good score (5.42). It is even sanctioned by a minimum score of 2.84. Via this rating, the message is conveyed that work is needed to improve the quality of the interface.

Furthermore, despite a higher score of 5.65 (lower by 1.25 when compared to *SysUse*), the global parameter *OVERALL* scores well showing a gap of 1.15 as compared to the average. This shows a slight compromise as to the overall satisfaction of interviewees on the functionalities of *Dialog Editor* as a whole.

2. Statistics by question

At the outset, it should be pointed out that an overview of the replies to the questions presents a certain disparity because, for practical reasons, the respondents replied N/A as a rating for certain questions in order to express an ability to decide.

Table 12. Cumulative responses assessments by query

Question	I strongly disagree	I disagree	I am so-so	I agree	I strongly agree
Q1	0	2	5	9	4
Q2	0	5	5	4	3
Q3	6	8	2	2	1
Q4	2	6	4	4	3
Q5	2	5	8	2	0
Q6	3	4	7	0	4
Q7	9	4	2	0	1
Q8	3	11	2	3	0
Q9	4	6	5	1	2
Q10	3	5	2	3	2
Q11	2	8	1	1	4
Q12	3	8	5	1	2
Q13	2	5	6	0	2
Q14	3	8	3	3	1
Q15	6	4	0	2	5
Q16	4	10	3	1	1
Q17	3	8	5	1	1
Q18	5	2	3	4	1
Q19	0	1	4	12	3

The analysis of the replies question by question (cfr Table 12) reveals a poor score for question 7 where 13 respondents clearly disagree. In other terms, the respondents are expressing certain difficulties understanding or using some concepts of *Dialog Editor*.

These poor scores serve to underline the need for, and importance of a basic level of knowledge to assist the users of *Dialog Editor*.

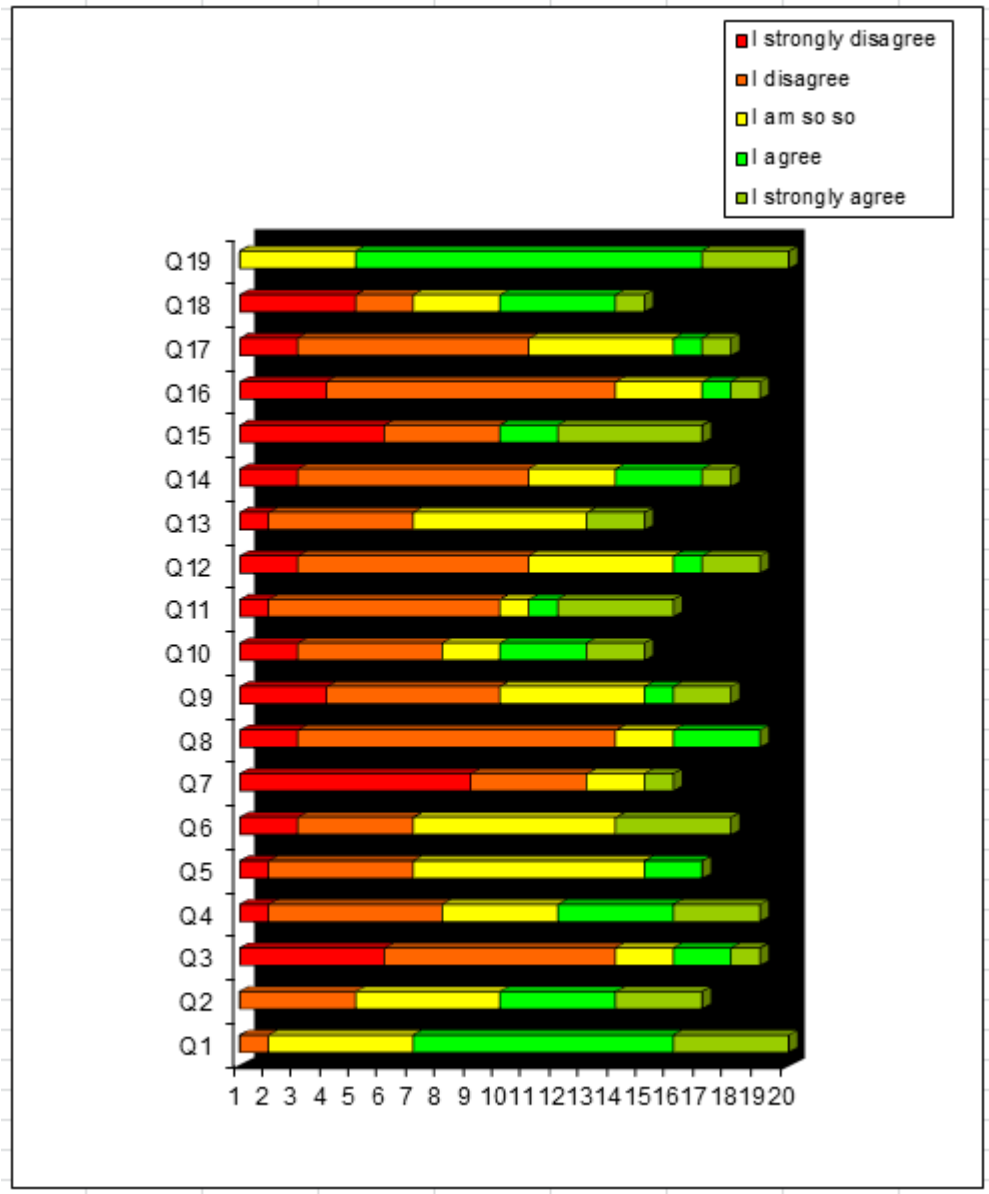


Figure 67. Queries' cumulative assessments

On the other hand, the best score is for the first question where only two people did not give approval to *Dialog Editor*, the others approving it particularly for its functionalities, such as code generation.

This score converges with the results obtained with the interviews. In fact, the code generation module that is more successful. Indeed, the gain is directly observable by users for whom the encoding of programs would be replaced by the specification via the editor scripts. Unfortunately, this task raises concerns. It is indispensable that the knowledge base plays its role to dispel doubt and encourages users to learn scripting

language. Suggestions and recommendations for extensions to implement the dialog editor to make attractive are listed in the general conclusion.

Table 13. Per question statistics

Query	Mean	Median	Average of deviations	Standard deviation
Q1	5,75	6	0,72	0,91
Q2	5,29	5	0,93	1,10
Q3	4	4	0,94	1,37
Q4	4,89	5	1,17	1,52
Q5	4,58	5	0,71	0,87
Q6	4,66	5	1,29	1,78
Q7	3,56	3	1,00	1,36
Q8	4,21	4	0,73	1,03
Q9	4,44	4	1,04	1,33
Q10	4,53	4	1,37	1,72
Q11	4,75	4	1,31	1,57
Q12	4,42	4	1,01	1,34
Q13	4,6	5	0,96	1,29
Q14	4,5	4	0,94	1,15
Q15	4,58	4	1,75	1,97
Q16	4,10	4	0,79	1,19
Q17	4,33	4	0,85	1,14
Q18	4,46	5	1,37	1,60
Q19	5,85	6	0,56	0,75

Statistics by questions presented in the Table13 confirm the analysis made previously. Overall, respondents are in agreement about the usefulness and usability of the Dialog Editor. However, we note a certain dispersion of views on certain questions. The Figure 68 below illustrates this fact better.

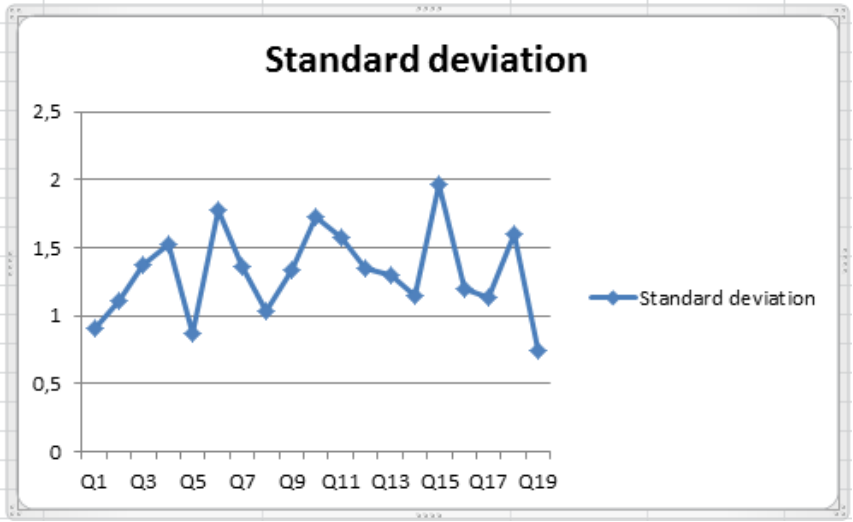


Figure 68. Queries' standard deviation

Question 15 has a high enough standard deviation (1.98). Let us note that 10 users disagree *Dialog Editor* quality information while 9 respondents approve this parameter. The explanation for this difference lies in the profiles of the respondents. Indeed, people who have a weak background in programming attach more importance to *Dialog Editor* content. Conversely, programmers are interested in its functions.

3. Conclusion

As we said in the Introduction, the Dialog Editor is at the prototype stage. It is in use with a single objective: to demonstrate the feasibility of the proposed methodology. However, although not completed, analysis of the satisfaction survey, via the CSUQ, shows that it is acceptable and that improvements will render it more useful and pleasanter to use.

5.3 Applying ISO/IEC 9126 Sofware Engineering

The purpose of this subsection is to examine the usefulness and the usability of the Dialog Editor by applying essentially criteria of the ISO/IEC 9126 software engineering.

It should be noted that the ISO/IEC 9126 does not provide requirements for software. However, it provides a framework for the evaluation of software quality. In other words, this standard defines a quality model which is applicable to every kind of software.

Let us specify that the model of the ISO/IEC 9126 classifies software quality in a structured set of six characteristics as shown in Figure 69.

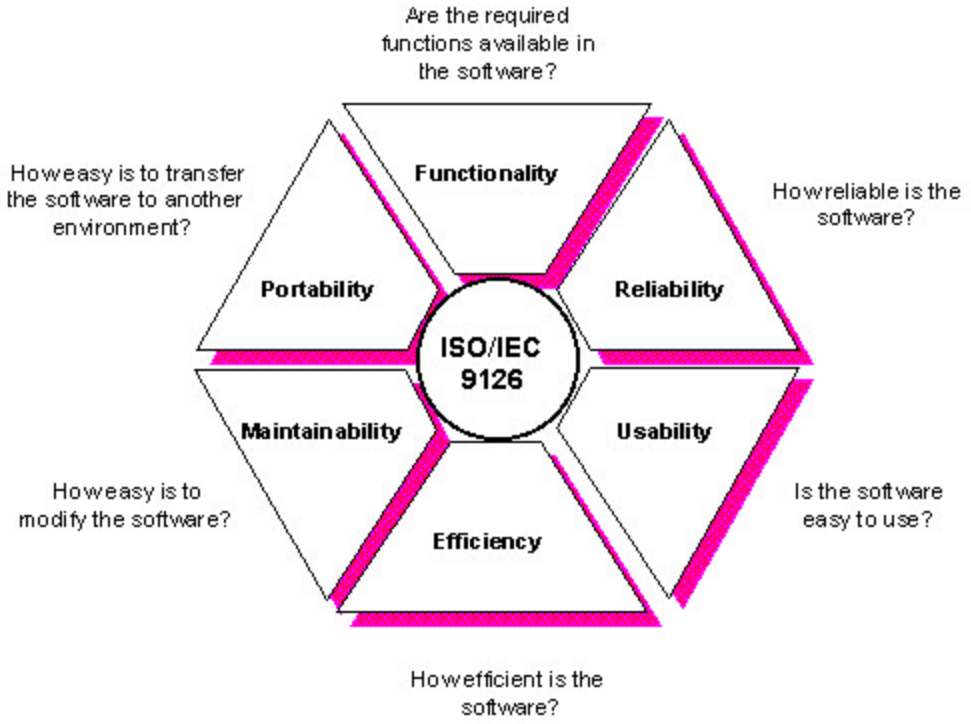


Figure 69. The six quality characteristics of a software.

We organize the rest of this chapter by drawing on the structure of the ISO9126. We will visit the *Dialog Editor* based on (sub-) characteristics identified in the ISO 9126 model. It should be noted that the editor that we will examine / critique is designed with an explorative aim to show, by implementation, the practical face of our conceptual model. Currently, many elements are missing or incomplete in order to make the *Dialog Editor* a production tool. Objectively, we will comment on each (sub) characteristic relative to the ISO9126 Model

5.3.1 Functionality

We confirmed previously that the implemented *Dialog Editor* is not a production tool. Nevertheless, we have planned, if not introduced, all necessary functions for the treatment of the methodology that we have built.

Nevertheless, we had to test the software extensively during the development of the CTI application. Admittedly, it still lacks some features but interestingly ever in its current state, it can already serve many purposes.

Suitability
Whether for project management, handling of mapping, the edition of dialogue scripts or code generation, the Editor offers the developer the functionality needed to carry out its task of specifying dialogues. Each function in the *Dialog Editor* engine contributes to implement one or more steps of the methodology algorithm.

Accuracy
There is a problem of accuracy with the scripting language. Indeed, as we do not define precisely the syntax and semantics of the generic language, the choice is left to the developer to clarify his instructions. The generic language implementation will be a challenge for the extension of this research.

Interoperability
The Editor is programmed with VB6 then, with *Installshield* software, we have prepared an installation file that runs perfectly under Mac OSX and Windows 32 bits. However, this file must be adapted to run under 64 bit.

Security
Taking into account the sensitivity of dialogue scripts, the Editor records scripts automatically every time the developer move from a tree node to another. For future work, it would not be luxury to extend this functionality to other tasks.

Functionality Compliance
We are not aware of a standard governing the functionality development of this type of software. Nevertheless, we relied on our long experience as a developer in the choice of icons; functions and user interface behaviour so that everything is intuitive for the designer.

5.3.2 Reliability

The few tests we have done can confirm the reliability of the *Dialog Editor*. For example, we had no crash during the implementation of case studies. Similarly, interviews were conducted without major problem during *Dialog Editor* demonstrations.

Maturity
It would be pretentious to affirm that the tool is mature, given that we are the only ones who exploit it *extensively*. With the note below, this software will reach maturity when, on the one hand, it will be fully programmed and, secondly, it will be extensively tested by users of different profiles.

Fault Tolerance
Despite many tests, no loss of information is to be reported due to a handling error. Admittedly, the source code was used directly. Under these conditions, any errors

occurring caused an interruption in the program, offering the possibility for correction and continuing.

Recoverability
We use XML files to keep information about objects, scripts and projects. There is no problem to recover data after an error.

Reliability Compliance
Taking into account the multiple examples we have achieved and, in particular, the big development that we have to implement CTI, enormous efforts have been made in error handling. Thus, we can definitely guarantee that the software is clearly well-qualified to pass the reliability test.

5.3.3 Usability

To make video clips that we posted on YouTube, we need voice of Valerie Bryce, an Englishwoman living in Belgium. Certainly, she uses a computer in her work but she has no background in programming or in user interface design.

During the working sessions we had with her, we were pleasantly surprised by her ability to understand concepts of the Dialog Editor. Without any doubt, her skills in the adaptation of new materials contributed significantly. But we also believe that the simplicity of *Dialog Editor* and logical connections between different concepts of the methodology facilitated her understanding.

With the example of Valerie and the reaction of different people who attended a demonstration of the Dialog Editor, we are more than convinced that the language and the logic of the *Dialog Editor* are not complicated to understand.

Otherwise, interviews and satisfaction surveys confirm that Dialog Editor is acceptable and that improvements will render it more useful and pleasanter to use. Indeed, let us remember and repeat that out of four general parameters of *IBM CSUQ*, *Sysuse* is the one obtaining the best score (6.09). However, there is a gap of 1.42 when compared to the average. This means that opinions are divided as far as the usefulness of *Dialog Editor* is concerned. Nevertheless, with an average of 4.67 (out of 7) we can conclude that the interviews approved the functionalities of Dialog Editor. The *Interqual* parameter is that which obtains the least good score (5.42). It is even sanctioned by a minimum score of 2.84. Via this rating, the message is conveyed that work is needed to improve the quality of the interface. Furthermore, despite a higher score of 5.65 (lower by 1.25 when compared to *SysUse*), the global parameter *OVERALL* scores well showing a gap of 1.15 as compared to the average. This shows a slight compromise as to the overall satisfaction of interviewees on the functionalities of *Dialog Editor* as a whole.

Similarly, the attempt of applying some early usability evaluation metrics proposed in [Pan08] on the main interface of dialogue scripting leads to the same observations. The last column of the table below lists a few remarks on the usability of the main scripting interface of Dialog Editor.

METRIC		SCORE	INDICATOR	REMARK
Title Length		12	Very Bad	This title is too long; it shows both the software and the user the current module.
Number of Font Style Used		3	Medium	There are three styles in the *RichTextBox* containing dialogue script
Word Number		10	Very Good	The few information messages or confirmation is very short and does not exceed 10 words.
Minimal Action		3	Good	Select a tree node, Edit interactive object Properties or Edit Script
Navigational Breadth	Tree of Objects	5	Good	Maximum Length: VB Projects Group→ VB Project → VB Form → VB Object Type → VB Instance Object
	grid of properties	2	Very Good	The grid is described by two columns; property name and the corresponding value.

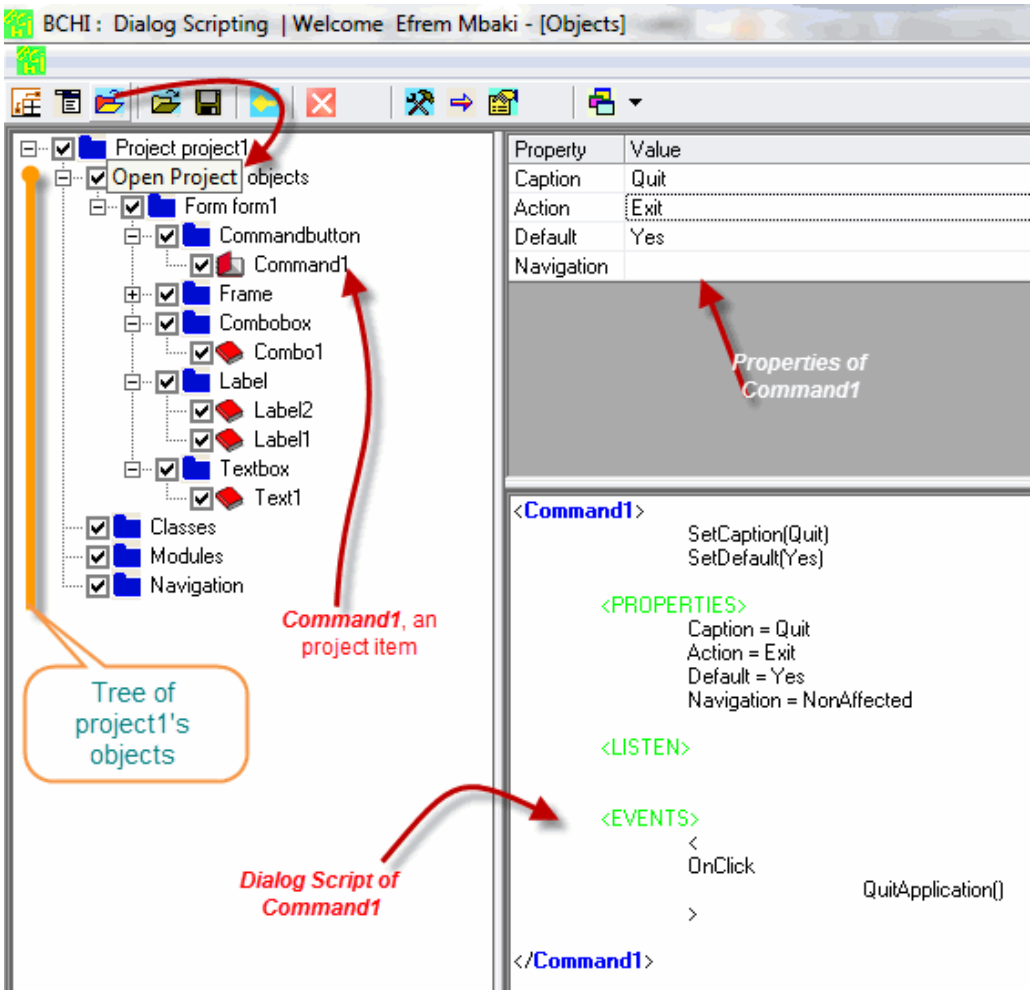


Figure 70. Dialog Scripting Interface

5.3.4 Efficiency

Our tool would not be interesting for one-shot software. Indeed, we should need to specify the interface, the mapping and the script before generating a code with high-level language, resulting in lost time and resources on things that may never re-used.

Let us consider that we are assuming the application to be processed is likely to evolve and operate in different contexts. Once the specification is complete, the developer will program any code or, if necessary, very few lines of codes. Indeed, the generator function will provide the executable program in the desired language. Although this is not the goal, ideally, an apprentice can manage an interactive project if he learns to use interactive editor.

Time Behaviour
Time lost in the specification is quickly gained during successive evolutions of the system. The project editor offers useful features for the efficient management of task extensions.
Currently, the only weakness of the editor is that the localization properties of geographic objects are set manually. In the future, these properties will be determined systematically using drag and drop.

Resource Utilisation
The *Dialog Editor* is very light. In addition, we chose to work with record sets and save the information in xml files. Even for large CTI application, the resources were barely noticeable.

5.3.5 Maintainability

Let us recall that the software architecture is organized in four modules completely independent of each other.

In addition, each module is structured in three parts: its interface, its functional machine and its dialogues controller. Under these conditions, it is easy to locate and correct errors. Similarly, extensions are easily implemented in the sense that the impact to other modules is minimal.

Analyzability
Analysis of conceptual objects is made by the algorithm of the methodology and the semantic functions are deduced systematically from the flow diagram that supports the method.

Changeability
At the risk of repeating myself, the technical choices make programming changes easy, without having to be concerned about other modules that are not affected by the updates.

Stability
We do not notice any signs of instability. The different examples treated show a convergence of functions developed. The size of the project has no negative effect on the Dialog Editor.

Testability
The tests are done module by module. Following the methodology, we test the crossing of such a step at any other stage. Each difficulty or blocking requires corrections in the software.

5.3.6 Portability

We noted above that the software runs on Windows 32-bit and Mac OSX. We find errors when using this software under Windows 64-bit. It would be a challenge for future changes

5.3.7 Conclusion

Overall, the qualities of the *Dialog Editor* meet the criteria of "ISO / IEC 9126 Software". This will significantly improve if we can complete its implementation.

We notice that the *Dialog Editor* offers a series of facilities in the interface design and behaviour of an interactive task. Also, we have highlighted features that are missing and those that must be completed.

5.4 Conclusion

The main objective of this fifth chapter was consideration of the usefulness and usability of Dialog Editor. To achieve this goal, we used a qualitative approach with three levels an evaluation plan:
(1) interviews with an open questionnaire
(2) satisfaction survey with IBM Computer Usability Satisfaction Questionnaire
(3) a discussion of the evaluation based on the standard ISO /IEC 9126

Information from these three analyzes highlight some qualities of *Dialog Editor* and expected improvements. With a good score, users agree *Dialog Editor* usefulness. However, they are critical about quality of *Dialog Editor*. The next chapter, the general conclusion, proposes improvements to address concerns and / or comments from users.

Chapter 6 Conclusion

Before beginning this chapter, it is important to remember that the statement of our thesis is the application of Model-Driven approach for designing the behaviour of multi-platform user interfaces.

We should recall that the structure of this thesis is organized into five main chapters. Let us make a summary aimed at highlight the fundamental elements and results. We will achieve this activity in three stages: firstly, we will make an overall summary of the results; secondly, we will highlight future work before, thirdly, concluding with some final remarks.

6.1 Global view

Initially, we explored the literature to establish the definitions and general concepts. In this way, we state the purpose of our thesis as clearly as possible in Chapter 1. It was also dealt with the interest of addressing this subject and we defined its limitations. A document plan was proposed in the aimed at clearly showing the interconnections between the different chapters.

In a second step, in Chapter 2, we identified and discussed research and / or results for dialogues in the field of human-machine interaction. Indeed, several techniques and methods are used to specify the dialogue but it is still an open subject because no method covers all areas of activity. Particular interest was placed on abstract machines, on specification languages and UsiXML. The advantages and limitations of these three tools were highlighted.

Having in mind our objective, a comprehensive review of the literature led us to impose two assumptions:

(1) Firstly, we exploited levels of distribution of Cameleon Reference Framework (CRF). Indeed, at the Abstract level, there is no representation at all: it is a purely theoretical level. This level describes potential user interfaces independently of any interaction modality and any implementation technology. At the Concrete level, we needed to fix the context of use and interaction modalities. This level describes a potential user interface after a particular interaction modality has been selected. The Final User Interface is reached when the code of a user interface is produced from the previous levels. This code could be either interpreted or compiled.

(2) Secondly, we used Model Driven Architecture (MDA). This choice is justified by a desire to separate the functional constraints from the technical constraints. MDA is a kind of machine used to create a model and refine it until, ideally, achieving the product, such as source code. With MDA, we define the system functionality in a model independent from the platform using a specification language. The resulting specification is translated into a specific model to a platform to finally generate the code compiled for the platform.

Based on these two assumptions, Chapter 3 described the core of our thesis. Indeed, to achieve our objective, we opted for a methodological approach with three branches:

(1) *The method*: we constructed a systematic approach, an algorithm, to achieve dialogues for the interactive task. Operating with the hypothesis of CFR, each point in this algorithm is in one of three levels of specification. With the MDA hypothesis, each of these points is a model. We built an algorithm which manipulates models and with functions/operators that make it possible to move from one model to another while remaining at an abstract, concrete or final level. So, the dialogue modelled at the abstract user interface level can be reified to the concrete user interface level by model-to-model transformation that can in turn lead to code by model-to-code generation

(2) *The model*: to support the above algorithm, we proposed a model-based conceptual model in which each exploited model is a toolkit; a kind of box of objects whose syntactic and semantic properties furnish dialogue scripts. Toolkits are classified according to the levels of abstraction of the CRF: task and domain, abstract user interface, concrete user interface and final user interface.

(3) *The Implementation*: to support the method and model described above, we implement graphical software called Dialog Editor. Indeed, definite concepts are general but in order to validate results, we limited ourselves to supporting three programming languages: Visual Basic, HTML Applications (HTA) and Microsoft Visual Basic for Applications (VBA). Two computing platforms are addressed: Microsoft Windows and Mac OS X. In this way, the approach demonstrates the capabilities of the abstractions in order to cover multiple programming paradigms and computing platforms. Five levels of behaviour granularity are exemplified through a step-by-step methodology that is supported by a project editor, a mapping editor, a script editor and a code generator integrated into a single authoring environment called, we noted, Dialog Editor.

The ideal would have been to implement completely the Dialog Editor. However, because of its complexity, we feared to exceeding the scope of our research. So, we limited ourselves to specifying a global view and implementing some modules in order to show its feasibility.

In Chapter 4, using *Dialog Editor* we carried out the exercise of applying the methodology on simple examples in which we tried to emphasize the five dialogue granularities.

Moreover, we made investment of implementing fully a complex application, named CTI Application, intended to cover the activities of a company which is specialized in the international transfer of money and import express worldwide services, CTI application. If for some simple examples we had no worries, we soon were limited while programming the CTI application. Finally, we were forced to develop some modules manually. It would be different if the *Dialog Editor* was completely finished.

The last chapter before this conclusion, Chapter 5, serves as a mirror. Indeed, it focuses on the evaluation of the results proposed in the third chapter. To that end, we have opted for a qualitative approach. Our assessment plan was structured in three phases: interviews, satisfaction survey and the application of ISO/IEC 9126 with the aim of examining the characteristics of Dialog Editor. It was an opportunity to explain our technical choices. It was also a way to demonstrate the limits of Dialog Editor. This exercise has shown the interest in completing *Dialog Editor* programming in order to improve its score on the six criteria of ISO/IEC 9126.

6.2 Summary of results

6.2.1 Theoretical and conceptual contributions

The conceptual model appears in section 3.2.3 and is our main theoretical contribution. Entities in this scheme are the source of our theory (*Reference to Concern#3, lack of modelling*).

Indeed, in order to apply MDE techniques, we need to define a dialog model that is expressive enough to accommodate advanced dialogues at different levels of granularity and different levels of abstraction, while allowing some structured design and development of corresponding dialogue. The *Dialogue Editor* described in this thesis will rely on this conceptual model. For this purpose, our conceptual modelling consists of expanding ECA rules towards dialogue scripting (or behaviour scripting) in a way that is independent of any platform (*Reference to Concern#5, lack of multiple platform managing*). This dialogue scripting is structured according to a meta-model that is reproduced in Figure 34 that enables defining a dialogue at five levels of granularity (*Reference to Concern#2, lack of managing complexity*):

1. Object-level dialogue modelling: this level models the dialogue at the level of any particular object, such as a CIO or a AIO. In most cases, UI toolkits and IDEs come up with their own widget set with built-in, predefined dialogue that can be only modified by overwriting the methods that define this dialogue. Only low-level toolkits allow the developer to redefine an entirely new dialogue for a particular widget, which is complex;

2. Low-level container dialogue modelling: this level models the dialogue at the level of any container of other objects that is a leaf node in the decomposition. Typically, this could be a terminal AC at the AUI level or a group box at the CUI level in case of a graphical interaction modality;

3. Intermediary-level container dialogue modelling: this level models the dialogue at the level of any nonterminal container of objects that is any container that is not a leaf node in the container decomposition. If the UI is graphical, this could be a dialog box or the various tabs of a tabbed dialog box;

4. Intra-application dialogue modelling: this level models the dialogue at the level of top containers within a same interactive application such as a web application or a web site. It therefore regulates the navigation between the various containers of a same application. For instance, the Open-Close pattern means that when a web page is closed, the next page in the transition is opened;

5. Inter-applications dialogue modelling: since the action term of an ECA rule could be either a method call or an application execution, it is possible to specify a same dialogue across several applications by calling an external program. Once the external program has been launched, the dialogue that is internal to this program (within-application dialog) can be executed.

6.2.2 Methodological contribution

Flow diagrams set out in Section 3.1.4 correspond to the model above and define the methodology that we propose to specify the interface and dialogues of an interactive task (*Refence to Concern#1, lack of methodology*). This algorithm operates on three levels of CFR and is fully independent of platform at abstract and concrete levels (*Reference to Concern#4, lack of computing-independent*).

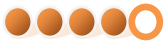
This thesis introduced an approach for conducting Model-Driven Engineering of dialogues for multi-platform GUIs that are compliant with the CRF. For this purpose, a Dialog Editor has been implemented that ultimately automatically generate code for four different targets (i.e., HTML V4.0, HTA, VBA V6.0, and DotNet V3.5) for two different computing platforms (Windows 7 and MacOS X) as a proof-of-concept. The main originality of this editor relies in its capability to always maintain a correspondence between native objects (belonging to the targets) and user objects (at AUI and CUI levels) and to support four types of mappings (i.e., forward, reverse, lateral, adaptation) possibly between two consecutive levels or not (cross-cutting). The Dialog Editor however only holds mappings for GUIs only, although interactive objects have been introduced for addressing Vocal User Interfaces (*Reference to Concern#4, lack of computing-independent*). Future work will be dedicated towards this goal and to integrate the conceptual model of dialogue into UsiXML V2.0 in an adequate way

6.2.3 Tools developed

We noted in the previous section we implemented of a software called *Dialog Editor* which is described in Section 3.3. Examples of the use of this tool are presented in Chapter 4. The criticism of this tool relative to interviews, *IBM CSUQ* and ISO/IEC 9126 criteria is described in Chapter 5.

The table below briefly describes the different modules of the dialog editor indicating its advantages and disadvantages. With five levels (1 the lowest and 5 the highest), we used three indicators (functional coverage, the index of complexity and stability of the code), to mention the current state of the editor of dialogue.

Table 14: Current State of Dialog Editor

MODULE	DESCRIPTION	(DIS)ADVANTAGES	MAIN IMPROVEMENTS	INDICATORS
Project Management	The Project Editor module includes all facilities required to create, retrieve, update and delete any UI project during the development life cycle. The Project Editor serves as the liaison between theoretical and practical scripts. The Project Editor may be assigned other tasks as necessary.	**Advantages** There is a single interface that provides all features for editing a project. **Disadvantage** The lack of sophistication, such as the positioning of objects in a container, significantly increases the difficulty of learning to use this tool.	1. Currently, we manage the opening of VB6, VBA and HTML files (projects) to enumerate all components (graphical or not). This module should be supplemented with the aim of supporting .NET files and others. 2. Integrate other models (abstract, concrete or final). For example, UsiXML models. 3. Implement a system that would automatically detect the geographic objects. For example, an interface where objects would be manipulated by dragging or pasting.	*Functional Coverage*  *Complexity Index*  *Code Stability* 
Script Editor	Script Editor is a text editor is specifically designed for developers to write the dialogue script of an interactive application or a program. This module is responsible for all services related to dialogue scripting. Scripts of some structured dialogues are automatically generated. In gen-	**Advantages** If knowledge of the scripting language is assumed, script editor is intuitively easy to use. In addition, without strict rules, the designer has complete freedom in managing dialog scripts. Also, some dialogue scripts are automatically generated from the properties of interactive objects **Disadvantage** Knowledge of the	1. Define the lexicon, the syntax and the semantics of the scripting language. 2. Objects and their properties are set in xml structures. Extending the Editor to allow dynamic property encoding; 3. Adding useful features, which may include colour syntax highlighting, auto in-	*Functional Coverage*  *Complexity Index* 

	eral, the designer has complete control over the operation of scripting. She/he can change the properties of the object dialogue in order to deduce partially or fully the behaviour of the object. Similarly, according to his will, he could manually write each line of code to set the behaviour of the current object.	scripting language is mandatory. In addition, the fact that the editor is not WYSIWYG could lead to difficulties in intuitively representing its objectives.	dentation, auto complete, bracket matching, syntax check, plugins, etc., to effectively support the users during coding, debugging and testing.	*Code Stability* 
Mapping Editor	This module is responsible for coordinating the transfer of a dialogue script from one abstract level to another as well as the controlling of schedules, transmitting proof, maintaining dialogue properties, status reports, etc. The Mapping Editor supports three types of mappings (i.e., forward, reverse, lateral) with the possibility to cross between two consecutive levels (cross-cutting). It uses the power of regular expressions to manage the relationship one-to-many	**Advantages** This tool allows the export of a project from one abstraction level to another, from one platform to another and from one model to another. It helps, for example, to finalize an interactive application with little programming knowledge. **Disadvantages** Mapping Editor proceeds object by object. And for a given object, it works with properties. Under such conditions, the volume may be difficult to manage. Supporting the complexity is not a strong point of this tool.	Conceptually, this module is finished. It still needs to be extended by including new models, objects and properties. In practice, it should support the mapping script dialogues. Indeed, the transformation of interactive objects is done.	*Functional Coverage*  *Complexity Index*  *Code Stability* 

Code Generator	At final level, the code generator translates from generic scripting to specific language relative to a target model. For objects whose properties exist in the system, the code generation is programmed in VBA, VB6 and HTA.	**Advantages** It should be noted that some of these scripts are automatically deduced through some attribute values. Other scripts are derived semi-automatically. Indeed, by combining the event of an interactive object to a function call, the developer will need to make the links between the function parameters (input and output) with the attributes of interactive objects. Then, the Editor automatically builds the script. **Disadvantages** As the majority of dialogue scripts are not generated automatically, it is necessary for the designer to have a good knowledge of the scripting language.	1. The completeness of interactive objects properties will allow the implementation of missing functions and the support of other toolkits and/or programming languages. 2. Construct a series of examples that could serve as patterns for best learning of the code generator module	*Functional Coverage*  *Complexity Index*  *Code Stability* 

6.3 Future work in prospect

In the near future, two extensions can be made to the results of this thesis, the first is theoretical and the second practical.

Indeed, firstly, it would be useful to apply programming language theory in order to define precisely the language of dialogue specification. The lexicon, the syntax and the semantics of this language are to be described. Then, it will follow the implementation of the interpreter of this language and its integration into the Dialog Editor.

In a second step, we will take the time to complete the programming of Dialog Editor. While respecting its architecture, functions will be taken one after the other to complete all the modules. Depending on need, we will consider the possibility of adding one or two targets platforms in the actual list.

In the medium term, we will consider investigating opportunities to develop mechanisms and gateways between UsiXML and the language of Dialog Editor. The table 8 above summarizes what is done and what remains to be done for the Dialog Editor.

References

A

[Acc96]

Accot, J., Chatty, S., and Palanque, P., *A formal Description of low level interaction and its application to multimodal interactive systems*, in Proc. of Eurographics Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'96 (Namur, June 1996), Springer, Berlin, 1996, pp. 92-104.

[Abr99]

Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S., and Shuster, J., *UIML: An Appliance-Independent XML User Interface Language*, in A. Mendelzon (ed.), Proceedings of 8th International World-Wide Web Conference WWW'8 (Toronto, May 11-14, 1999), Elsevier Science Publishers, Amsterdam, 1999. Accessible at http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html.

[Ari88]

Ariav, G. and Calloway, L.-J., *Designing conceptual models of dialogue: A case for dialogue charts*, SIGCHI Bulletin, vol. 20, no. 2, 1988, pp. 23–27.

B

[Bas99]

Bastide R. and Palanque P., *A Visual and Formal Glue Between Application and Interaction*, Journal of Visual Language and Computing, 10(5), October 1999, pp. 481–507.

[Ber01]

Berstel, J., Reghizzi S.C., Roussel G. and Pietro P.S., *A scalable formal method for design and automatic checking of user interfaces*, Proc. of 23rd Intl. Conf. on Software Engineering ICSE 2001 (Toronto, 12-19 May 2001), IEEE Computer Society, Los Alamitos, 2001, pp. 453–462.

[Bez04a]

Bézivin J., *In Search of a Basic Principle for Model Driven Engineering*, UPGRADE the European Journal for the Informatics Professional, Vol. V, No. 2, April 2004

[Bez04b]

Bézivin J., *Model Engineering for Software Modernization*, The 11th IEEE Working Conference on Reverse Engineering, Delft, November 8th-12th 2004.

[Bez05]

Bézivin J., On the unification power of models, Software and Systems Modeling 4(2): 171–188, May 2005.

[Bod95]

Bodart F., Hennebert A.-M., Leheureux J.-M., Provot I., Vanderdonckt J. and Zucchinetti G., *Key Activities for a Development Methodology of Interactive Applications*, Chapter 7, in Benyon, D., Palanque, Ph. (Eds.), "Critical Issues in User Interface Systems Engineering", Springer-Verlag, Berlin, 1995, pp. 109-134.

[Bod00]

Bodart, F., Leheureux, J.-M., Mbaki, E., Vanderdonckt, J.,Windows Transitions: *A Graphical Notation for Specifying Mid-Level Dialogue Models*, Informal Proc. of 7th Int. Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2000 (Limerick, 5-6 June

2000), Ph. Palanque, F. Paternò (eds.), 2000.

[Boo04]

Book M. and Gruhn V., *Modeling web-based dialogue flows for automatic dialogue control*, Proc. of 19th IEEE Intl. Conf. on Automated Software Engineering ASE'2004 (Linz, 20-25 September 2004), IEEE Computer Society, Los Alamitos, 2004, pp. 100–109.

[Boo05a]

Book M. and Gruhn V., *Experiences with a dialogue-driven process model for web application development*, Proc. of 29th Annual Intl. Computer Software and Applications Conf. COMPSAC'2005 (Edinburgh, 25-28 July 2008), IEEE Computer Society, Los Alamitos, 2005, pp. 173–178.

[Boo05b]

Book M., Gruhn V. and Mirbach N. *A Meta-Model for the Dialogue Flow Notation*, Proc. of the 1st Intl. Conf. on Web Information Systems and Technologies WE-BIST'2005 (Miami, 26-28 May 2005), INSTICC Press, 2005, pp. 64–71.

[Boo06a]

Book M. and Gruhn V., *Efficient Modeling of Hierarchical Dialogue Flows for Multi-Channel Web Applications*, Proc. of 30th Annual Intl. Computer Software and Applications Conf. COMP¬SAC'2006 (Chicago, 17-21 September 2006), IEEE Computer Society, Los Alamitos, 2008, pp. 161–168.

[Boo06b]

Book M. and Gruhn V and Lehmann M., *Automatic dialogue mask generation for device-independent web applications*, Proc. of 6th Intl. Conf. on Web Engineering ICWE 2006 (Palo Alto, 11-14 July 2006), ACM Press, New York, 2006, pp. 209–216.

[Boo07]

Book M. and Gruhn V. and Richter J., *Fine-grained specification and control of data flows in web-based user interfaces*, Proc. of 7th Intl. Conf. on Web Engineering ICWE'2007 (Como, 16-20 July 2007), Lecture Notes in Computer Science, Vol. 4607, Springer-Verlag, Berlin, 2007, pp. 167–181.

[Boo08]

Book M. and Gruhn V, *Efficient Modeling of Hierarchical Dialogue Flows for Multi-Channel Web Applications*, Proc. of 30th Annual Intl. Computer Software and Applications Conf. COMP¬SAC'2006 (Chicago, 17-21 September 2006), IEEE Computer Society, Los Alamitos, 2008, pp. 161–168.

[Bre09]

Breiner, K., Maschino, O., Görlich, D., Meixner, G. *Towards automatically interfacing application services integrated in a automated model based user interface generation process*. In Proc. Of MDDAUI'2009 at at http://ceur-ws.org/Vol-439/paper5.pdf, Sanibel Island, Florida, USA, febuary 2009

[Bri87]

Britts S., *Dialogue management in interactive systems: a comparative survey*, SIGCHI Bulletin, 18(3), 1987, pp. 30–42.

[Bro07]

Brossard A., Abed M., Kolski C. *Modélisation conceptuelle des IHM. Une approche globale s'appuyant sur les processus métier*. Ingénierie des Systèmes d'Information 12(5): 69-108 (2007)

[Bro09]

Brossard A., Abed M., Kolski C. and Uster G. *User modelling: the consideration of the experience of time during journeys in public transportation*. Mobility Conference 2009, Proceedings of the 6th International Conference on Mobility Technology, Applications and Systems (2-4 september, 2009, Nice), ACM Press

[Bro11]

Brossard A., Abed M., Kolski C., *Taking context into account in conceptual models using a Model Driven Engineering approach*, Information and Software Technology, volume 53, pp. 1349–1369, 2011

C

[Cac03]

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. *A Unifying Reference Framework for Multi-Target User Interfaces*. Interacting with Computers 15, 3 (2003) 289–308.

[Cac07]

Cachero C., Melia S., Poels G. and Calero C. , *Towards improving the navigability of Web Applications: a model-driven approach*, European Journal of Informations Systems, 16, 2007, pp. 420–447.

[Cal05]

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. *A Unifying Reference Framework for Multi-Target User Interfaces*. Interacting with Computer 15,3 (2003) 289–308.

[Can05]

Cantera, J.M., González Calleros, J.M., Meixner, G., Paternò, F., Pullmann, J., Raggett, D., Schwabe, D., Vanderdonckt, J. *Model-Based UI XG Final Report. W3C Incubator Group Report*, 4 May 2010. Available at: http://www.w3. org/2005/Incubator/model-basedui/XGR-mbui/8. Carr, D. Specification of interface interaction objects.In Proc. of CHI'94. ACM Press, New York, 1994.

[Car94]

Carr D.A., *Specification of interface interaction objects*, Proc. of ACM Conf. on Human Aspects in Computing Systems CHI'94 (Boston, 24-28 April 1994), ACM Press, New York, 1994, pp. 372–378.

[Car98]

Carpenter, R. H. S., & Robson, J. G. (Eds.). (1998). *Vision research: A practical Guide to Laboratory Methods*. Oxford, England: Oxford University Press.

[Cle06]

Clerckx, T., Van den Bergh, J. and Coninx, K. *Modeling Multi-Level Context Influence on the User Interface*. In Proc. of PERCOMW'2006. IEEE Press, 2006, pp. 57–61.

[Cow95]

Cowan D. and Pereira de Lucena C., C. *Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse*, IEEE Transactions on Software Engineering, 21(3), 1995, pp. 229-243.

[Dit04]

Dittmar, A. and Forbrig, P. *The Influence of Improved Task Models on Dialogues*. In Proc. of CADUI'2004, pp. 1–14, Kluwer Academic Publishers, 2005, Netherlands

[Dix98]

Alan J. Dix, Janet E. Finlay, Gregory D. Abowd, Russell Beale, *Human Computer Interaction*, Prentice Hall, 01/12/1998

[Duc07]

Duchowski, A. T. (Ed.). (2007). *Eye Tracking Methodology*: Theory and Practice (2nd ed.). New York: Springer-Verlag.

E

[Eise01]

Eisenstein, J., Vanderdonckt, J. and Puerta, A., *Applying Model-Based Techniques to the Development of UIs for Mobile Computers*. In IUI01: 2001 International Conference on Intelligent User Interfaces. pp. 69-76, Santa Fe, NW.

[Elw96]

Elwert, T. *Continuous and Explicit Dialogue Modelling*. In Proc. of EA-CHI'96. 265 - 266 ACM, 1996, New York, USA

G

[Gat08]

Gates B., Bill Gates Keynote: *Microsoft Tech-Ed. 2008 – Developers*. <http://www.microsoft.com/presspass/exec/billg/speeches/2008/06-03teched.mspx>.

[Goe96]

Goedicke M. and Sucrow B.E., *Towards a formal specification method for graphical user interfaces using modularized graph grammars*, Proc. of the 8th Intl. Workshop on Software Specification and Design IWSSD'96 (Washington, DC, 1996), IEEE Computer Society, Los Alamitos, 1996.

[Gom01]

Gomez J., Cachero C., Pastor O., *Conceptual Modeling of Device-Independent Web Applications*, IEEE Multimedia, 8(2), 2001, pp. 26-39.

[Gre86]

Green M., *A survey of three dialogue models*. ACM Trans. on Graphics, 5(3), Juy 1986, pp. 244–275.

H

[Hako10]

Håkon Wium L and Bert B., *Cascading style sheets: designing for the Web*. Addison Wesley Longman. p. 263. Retrieved 9 June 2010.

[Han06]

Han M. and Hofmeister C., *Modeling and verification of adaptive navigation in web applications*. Proc. of 6th Intl. Conf. on Web Engineering ICWE 2006 (Palo Alto, 11-14 July 2006), ACM Press, New York, 2006, pp. 329–336.

[Han03]

Hansmann U., Merk L., Nicklous M., Stober T., *Pervasive Computing: The Mobile World*, second ed., Springer Professional Computing, 2003.

[Har87]

Harel D., *Statecharts: A visual formalism for complex systems*, Science of Computer Programming, 8, 1987, pp. 231–274.

[Hay86]

Hayes Ian J. *Using mathematics to specify software* In Proceedings of the 1st Australian Software Engineering Conference, ASWEC-86., May 1986, pp. 67–71.

[Hel09]

Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., Vanderdonckt, J. *Human-Centered Engineering with the User Interface Markup Language*. In "Human-Centered Software Engineering", Chapter 7, HCI Series, Springer, London, 2009, pp. 141–173.

[Hil86]

Hill R.D., *Supporting Concurrency, Communication and Synchronization in Human-Computer Interaction - The Sassafras UIMS*, ACM Transactions on Graphics, 5(3), 1986, pp. 179-210.

[Hus99]

Hussey A. & Carrington D. *Model-Based Design of User-Interfaces using object-Z*. In CADUI'99 Computer-Aided Design of User Interfaces II by Vanderdonck J & Puerta A., Kluwer Academic Publishers, pp. 43-53. ACM Press, 1999.

I

[Iso01]

ISO/IEC 9126-1:2001 *Software engineering — Product quality — Part 1: Quality model*

[Iso11]

ISO/IEC 25010:2011, *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models*

J

[Jac86]

Jacob, R.J.K. *A specification language for direct manipulation user interfaces*. ACM Transactions on Graphics, 5, 4 (1986) 283–317.

[Jan93]

Janssen C., Weisbecker A. and Ziegler J., *Generating user interfaces from data models and dialogue net specifications*, Proc. of ACM Conf. on Human Aspects in Computing Systems InterCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 418–423.

[Jon80]

Jones Cliff B. *Software Development: A Rigorous Approach*. Prentice Hall International., 1980, ISBN 0-13-821884-6.(VDM)

[Jon86]

Jones Cliff B. *Systematic Software Development using VDM*. Prentice Hall International, 1986, ISBN 0-13-880717-5.

K

[Kle88]

Kleyn M.F. and Chakravarty I., *Edge - a graph based tool for specifying interaction*, Proc. of 1st Annual ACM Symposium on User Interface Software and Technology UIST'88 (Alberta, 17-19 October 1988), ACM Press, New York, 1988, pp. 1–14.

L

[Lew95]

Lewis J.R., *IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use*. International Journal of Human-Computer Interaction, 1995, 7 (1), 57-78

[Lik32]

Likert. R., *A technique for the measurement of attitudes*. Archives of Psychology, 22(140):1–55, 1932.

[Lim04]

Limbourg Q. and Vanderdonckt J., *Addressing the Mapping Problem in User Interface Design with UsiXML*, Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004), ACM Press, New York, 2004, pp. 155–163.

[Luy03]

Luyten K., Clerckx T., Coninx K. and Vanderdonckt J., *Derivation of a Dialogue Model from a Task Model by Activity Chain Extraction*, Proc. of 10th Int. Conf. on Design, Specification and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 203–217.

M

[Mba00a]

Mbaki, E., *Towards a Library of Generic Guidelines for Specifying Multi-Threaded Dialogs*. In, Vanderdonckt, J., Farenc, Ch. (Eds.), Tools for Working with Guidelines, Proc. of the Int. Workshop on Tools for Working with Guidelines TFWWG'2000 Group (Biarritz, 7-8 October 2000), Springer-Verlag, London, 2000. , pp. 217-224.

[Mba00b]

Mbaki Luzayisu E., *Utilisation des automates à pile pour la spécification de dialogues et la gestion de l'historique*. Actes de p.47-52, Actes des Rencontre de Jeunes Chercheurs en Interface Homme-Machine, RJC-IHM'2000, 3-5 mai 2000, Ile de Berder, Golfe du Morbillan, France,http://www-valoria.univubs. fr/RJCIHM/Actes/actes.htm

[Mba02]

Mbaki E. and Vanderdonckt J., *Window Transitions: A Graphical Notation for Specifying Mid-level Dialogue* In Proc. of 1st Int. Workshop on Task Models and Diagrams for user interface design Tamodia'2002 (Bucharest, 18-19 July 2002), Academy of Economic Studies of Bucharest, INFOREC Printing House, Bucharest, 2002, pp. 55–63.

[Mba08]

Mbaki, E., Vanderdonckt, J., Guerrero, J. and Winckler, M. *Multi-level Dialogue Modeling in Highly Interactive Web Interfaces*. In Proc. of IWWOST'2008, CEUR Workshop Proc., Vol. 445, 2008, pp. 38–43.

[Mba11a]

Mbaki, E., Vanderdonckt, J., *Model-Driven Engineering of Behaviors for User Interfaces in Multiple Contexts of Use*. In Proc. of IADIS Int. Conf. on Interfaces and Human-Computer Interaction IHCI'2011 (Rome, 24-26 July 2011), IADIS Press, Rome, 2011, pp. 273-282

[Mba11b]

Mbaki, E., Vanderdonckt, J., Winckler, M., *Model-Driven Engineering of Dialogues for Multi-platform Graphical User Interfaces*. In Proc. of 2nd Int. Workshop on User Interface Extensible Markup Language UsiXML'2011 (Lisbon, 6 September 2011), Thales Research and Technology France, Paris, 2011, pp. 169-180

[Mba99]

Mbaki Luzayisu E., *Modélisation et spécification des dialogues relatifs à des applications de gestion*, Tome II, Actes de IHM'99, du 23 au 26 novembre 1999, Montpellier, France

[Mei09a]

Meixner, G., Görlich, D., Breiner, K., Hußmann, H.,Pleuß, A., Sauer, S., Van den Bergh, J. S *Selecting the Right Task Model for Model-based User Interface Development*. In Proc. of 4th Int. workshop on model driven development of advanced user interfaces. MDDAUI'2009. In Proc. of IUI 2009, pp. 503–504.

[Men03]

Menkhaus, G. and Fischmeister, S. *Dialogue Model Clustering for User Interface Adaptation*. In Proc. of ICWE'2003. LNCS, Vol. 2722, Springer-Verlag, Oviedo, Spain, 2003, pp. 194–203.

[Mon05]

Montero F., López-Jaquero V., Vanderdonckt J., Gonzalez P., Lozano M.D. and Limbourg Q., *Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML*, Proc. of 12th Intl. Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'2005 (Newcastle upon Tyne, 13-15 July 2005), S.W. Gilroy, M.D. Harrison (eds.), Lecture Notes in Computer Science, Vol. 3941, Springer-Verlag, Berlin, 2005, pp. 161-172.

[Mor04]

Mori G., Paternò F. and Santoro C., *Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions*, IEEE Transactions On Software Engineering, Vol. 30, No. 8, August 2004, pp. 507-520

O

[Oli01]

Oliveira (de) M.C.F., Turine M.A.S. and Masiero P. C., *A statechart-based model for hypermedia applications*, ACM Transactions on Information Systems, 19(1), 2001, pp. 28–52.

[Ols84]

Olsen D., *Pushdown automata for user interface management*, ACM Transactions on Graphics, 3(3), 1984, pp. 177–203.

[Omg00]

Object Management Group and R. Soley. *Model-Driven Architecture*, 2000. OMG document available at www.omg.org.

[Omg05]

OMG, *QVT Final Adopted Spec.* , www.omg.org/docs/ptc/05-11-01.pdf, November 2005

[Omg08]

OMG, *QVT 2.0 Transformation Spec.*, http://www.omg.org/spec/QVT/1.0/PDF/, Avril 2008

P

[Pal94]

Palanque P. and Bastide R., *Petri net based design of user-driven interfaces using interactive cooperative object formalism*, Proc. of 1st Eurographics Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'94 (Bocca di Magra, June 1994), Springer Verlag, Vienna, 1994.

[Pat08]

Jose Ignacio Panach, Nelly Condori-Fernández, Francisco Valverde, Nathalie Aquino, Oscar Pastor, *Towards an Early Usability Evaluation for Web Applications*, Software Process and Product Measurement, Lecture Notes in Computer Science Volume 4895, 2008, pp 32-45

[Pat09]

Paternò, F., Santoro, C. and Spano, L.C. *MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments*. ACM Transactions on Computer-Human Interaction 16, 4 (November 2009), Article 19.

[Pat94]

Paternò F & Leonardi A, *a semantic-based approach for the design and implementation of interaction objects*, Computer Graphics Forum 13(3), 1994, pp.195-204.

[Pau99]

Paulo F.B., Masiero P.C. and de Oliveira M.C.F., *Hypercharts: Extended statecharts to support hypermedia specification*, IEEE Transactions on Software Engineering, 25(1), 1999, pp. 33–49.

[Pay86]

Payne and T.R.J Green, *Task-action grammars: A model of the mental representation of task languages*, Human-Computer Interaction, 2(2), 1986, pp. 93–133.

[Ple05a]

Pleuß, A. *Modeling the User Interface of Multimedia applications*. In Proc. of MoDELS 2005, pp. 676–690. ontego Bay, Jamaica

[Ple05b]

Pleuß, A. *MML: A Language for Modeling Interactive Multimedia Applications*. In Proc. of ISM'2005, pp. 465–473, Irvine (CA), USA

R

[Rai04]

Raistrick C.; Colin C., Paul F.; Ian W.; *John Wright. Model Driven Architecture with Executable UML*. Cambridge University Press, 2004.

[Rei08]

Reichart, D., Dittmar, A., Forbrig, P. and Wurdel, *M.Tool Support for Representing Task Models*, Dialogue Models and User-Interface Specifications. In Proc. of DSV-IS'2008. LNCS, Vol. 5136, Springer, Berlin, 2008, pp. 92–95.

[Rei81]

Reisner P., *Formal grammar and human factors design of interactive graphics systems*, IEEE Transactions on Software Engineering, 7, 1981, pp. 229–240.

[Rüc08]

Rückert, J. and Paech, B. *The Guilet Dialogue Model and Dialogue Core for Graphical User Interfaces*. In Proc. of EIS'2008. LNCS, Vol. 5247, Springer, 2008, pp. 197–204.

S

[Sch06]

Schmidt D.C, *Guest Editor's Introduction: Model-Driven Engineering*, Computer 39 (2006), pp. 25-31.

[Sch07]

Schaefer R., Bleul S., Müller W., *Dialogue Modeling for Multiple Devices and Multiple Interaction Modalities*, Proc. of 5th Int. Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2006 (Hasselt, 23-24 October 2006), K. Coninx, K. Luyten, K. Schneider (eds.), Lecture Notes in Computer Science, Vol. 4385, Springer-Verlag, Berlin, 2007, pp. 39–53.

[Sch95]

Schneiher K. & Ander Repenning A. *Deceived by Ease of Use*. In DIS'95 Symposium on Designing Interactive Systems: Processes, Practices, Methods an Techniques, pp. 177-188. ACM Press, 1995.

[Sch04]

Schaefer R., Bleul S., Müller W., *A Novel Dialogue Model for the Design of Multimodal User Interfaces*, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Work-shop on Design, Specification and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004), Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 221–223.

[Sch07]

Schaefer, R., Bleul, S. and Müller, W. *Dialogue Modeling for Multiple Devices and Multiple Interaction Modalities*. In Proc. of TAMODIA'2006. Lecture Notes in Computer Science, Vol. 4385, Springer-Verlag, Berlin, 2007, pp. 39–53.

[Sol00]

Soley R. and the OMG staff. *Model Driven Architecture*. November 2000, ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf

[Spi92]

Spivey J. Michael. *The Z Notation: A reference manual (2nd edition ed.)*. Prentice Hall International Series in Computer Science, 1992. ISBN 0-13-978529-9.(Z)

[Sta73]

Stachowiak H., *Allgemeine Modelltheorie*, Springer-Verlag edition, in German, 1973

[Sta07]

State Chart XML (SCXML): State Machine Notation for Control Abstraction. W3C Working Draft, 21 February 2007, http://www.w3.org/TR/scxml/

[Tra03]

Traetteberg, H. *Dialogue modelling with interactors and UML Statecharts*. In Proc. of DSV-IS'2003. LNCS, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 346–361.

[Tra08]

Traetteberg, H. *Integrating Dialogue Modeling and Domain Modeling – the Case of DIAMODL and the Eclipse Modeling Framework*. Journal of Universal Computer Science 14, 19 (2008), pp 3265–3278.

V

[Van03]

Vanderdonckt J., Limbourg Q., Florins M., *Deriving the Navigational Structure of a User Interface*, Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2003 (Zurich, 1-5 September 2003), M. Rauterberg, M. Menozzi, J. Wesson (eds.), IOS Press, Amsterdam, 2003, pp. 455–462.

[Van07]

Van den Bergh, J. and Coninx, K. *From Task to Dialogue model in the UML*. In Proc. of Tamodia'2007, pp. 98–111, Springer-Verlag Berlin, 2007

[Van98]

Vanderdonckt J., Tarby J.-Cl. and Derycke A., *Using Data Flow Diagrams for Supporting Task Models, Supplementary*. Proc. of 5th Int. Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'98 (Abingdon, 3-5 June 1998), P. Markopoulos, P. Johnson (eds.), Eurographics Association, Aire-la-Ville, 1998, pp. 1–16.

[Van99]

Van Welie, M., van der Veer, G.M.C. and Eliëns, A. *Usability Properties in Dialogue Models*. In Proc.

of DSV-IS'99, Springer-Verlag, Universidade do Minho, Braga, Portugal, 2-4 June 1999.

[Ver12]

Verhagena Wim J.C., Bermell-Garciab Pablo, Van Dijkc Reinier E.C., Curran Richard, *In Advanced Engineering Informatics archive*, Volume 26 Issue 1, January, 2012, Pages 5-15

[Vev96]

Vivier J., *Introduction : la psycholinguistique au secours de l'informatique*. In: Langages, 35e année, n°144, 2001. pp. 3-19**,** Paris, 1996.

W

[W3C08]

W3C State Chart XML (SCXML), *State Machine Notation for Control Abstraction*, Working Draft, 16 May 2008. Accessible at http://www.w3.org/TR/ SCXML.

[Was85]

Wasserman A., *Extending State Transition Diagrams for the Specification of Human-Computer Interaction*, IEEE Transactions on Software Engineering, 11(8), 1985, pp. 699–713.

[Win03]

Winckler M. and Palanque P.. *StateWebCharts: A formal description technique dedicated to navigation modelling of web applications*. Proc. of 10th Int. Conf. on Design, Specification and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 61-76.

[Win04]

Winckler M., Barboni E., Farenc C., Palanque P., *SWCEditor: a Model-Based Tool for Interactive Modelling of Web Navigation*. In Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2004 (Funchal, 14-16 January 2004), Kluwer Academic Publishers, Dordrecht, 2005, pp. 55–66.

[Win08]

Winckler M., Trindade F., Vanderdonckt J., *Cascading Dialogue Modeling with UsiXML*, Proc. of 15th Int. Work-shop on Design, Specification and Verification of Interactive Systems DSV-IS'2008 (Kingston, July 16-18, 2008), Lecture Notes in Computer Science, Springer, Berlin, 2008.

[Wir92]

Wirth N. and Gutknecht J., *Project Oberon- The design of an Operating System and Compiler*, Addison-Wesley, Reading, Mass. 1992

[Yam09]

Yamane K. and Hodgins J.K.: *Simultaneous Tracking and Balancing of Humanoid Robots for Imitating Human Motion Capture Data*, Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2510-2517, 2009

[Yam10]

Yamane K. and Hodgins J.K.: *Control-Aware Mapping of Human Motion Data with Stepping for Humanoid Robots*, Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 726-733, 2010

Annex A. Password Evaluation

By exploiting a simple example: evaluating a password, the purpose of this annex is twofold. At first, it illustrates some important concepts in the specification and design of an interactive task such as software architecture and the dialogue automata. Moreover, these lines are also of interest to show the advantages and disadvantages of working manually or getting help from the Dialog Editor

A.1. Statement

In this example, we try to evaluate a password. The formula that we decide to apply is very simple. Indeed, a password is better if, at the same time:

1. its length is between 6 and 15 characters. Shorter would be too easy to identify and, more length, more difficult to retain ;
2. it contains at least one lowercase letter;
3. it contains at least one uppercase letter;
4. it contains at least one numerical digit and
5. it contains at least a special character such as comma, semicolon,…

We suggest that the user has two modes of interaction. By using a command button, he/ can choose to view his password (to read a word such as it is) or to hide his password (to replace all the characters by *)

Each time the user modifies his password, an evaluation will be made. The result expressed as a percentage will be posted. In parallel, the user will see measures on a slide object at the right position. Lastly, the user has another command button which enables him to leave the interface.

A.2. Architecture and behaviour automata

As shown in the Figure 70, we propose a simple architecture with three components: the use interface, the functional machine function and the controller that manages information exchanges. The module initialization involves the preparation of useful resources for working.

For a better understanding, let us use automata (Figure 71) to illustrate the behaviour of our application. It is important to note that our machine needs to keep its internal state. For example, to return to hide or view state after the evaluation, the automata must remember the internal state of his departure. Otherwise, the behaviour we describe will not match the requested dialogue.

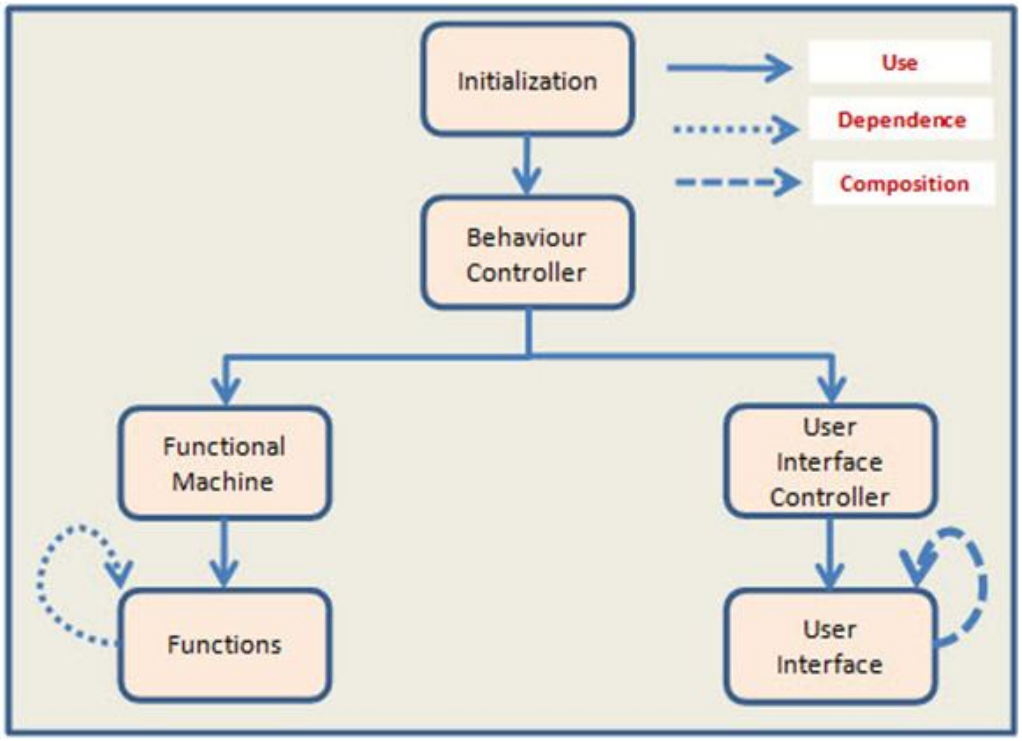


Figure 71. Global architecture.

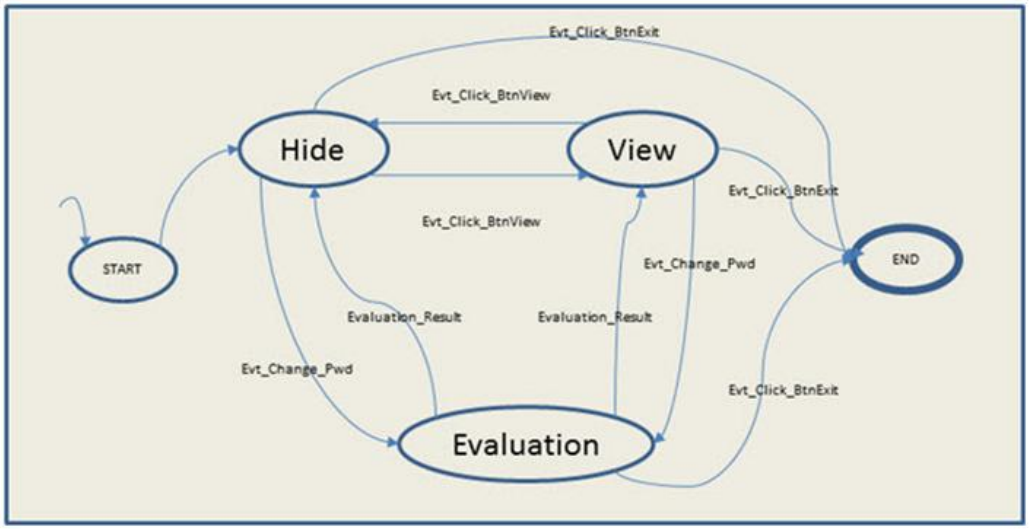


Figure 72. Dialogue Automata.

A.3. User interface

A.3.1. Global view

In Visual Basic 6, a User Interface answering the above specification could be built in the following way.

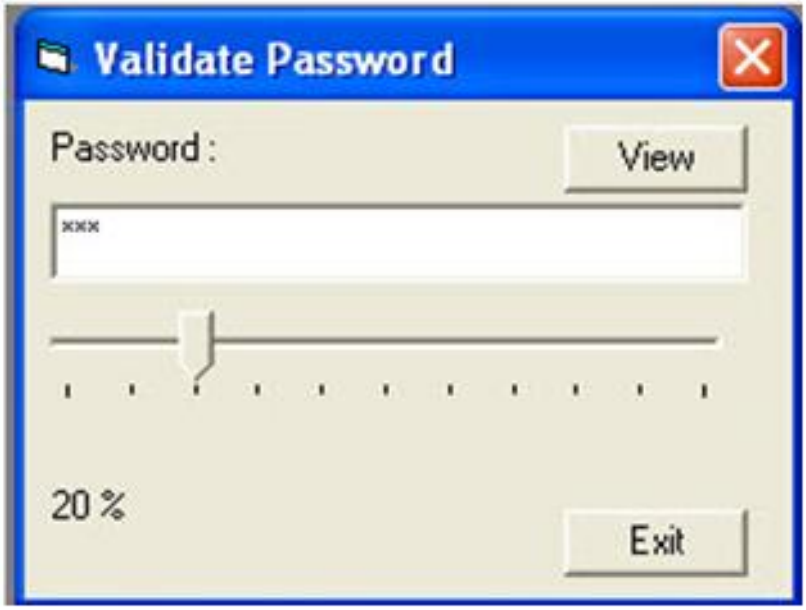


Figure 73. VB6 Password Interface.

The main Form is named *frmMain*. It contains two command buttons. The first one labelled "Exit", is used to leave the application. The second, labelled here "View", can have its caption changed to "Hide" with a click by a user. This command button helps the user to view/hide its password. The value of the Slide object indicates the evaluation result which is also marked by a label

A.3.2 Manual User Interface (with visual basic editor)

Manual building for this interface requires the launching of Microsoft Visual Basic 6.0 where we choose to create an exe application.

Figure 74. Visual Basic 6 IDE.

Automatically, the system offers a new empty window where we drag-and-drop general objects; two command buttons, two labels and a textbox. The developer could fix object properties with the mouse pointer and arrow keys or by typing values in the properties table shown in the Figure 73.

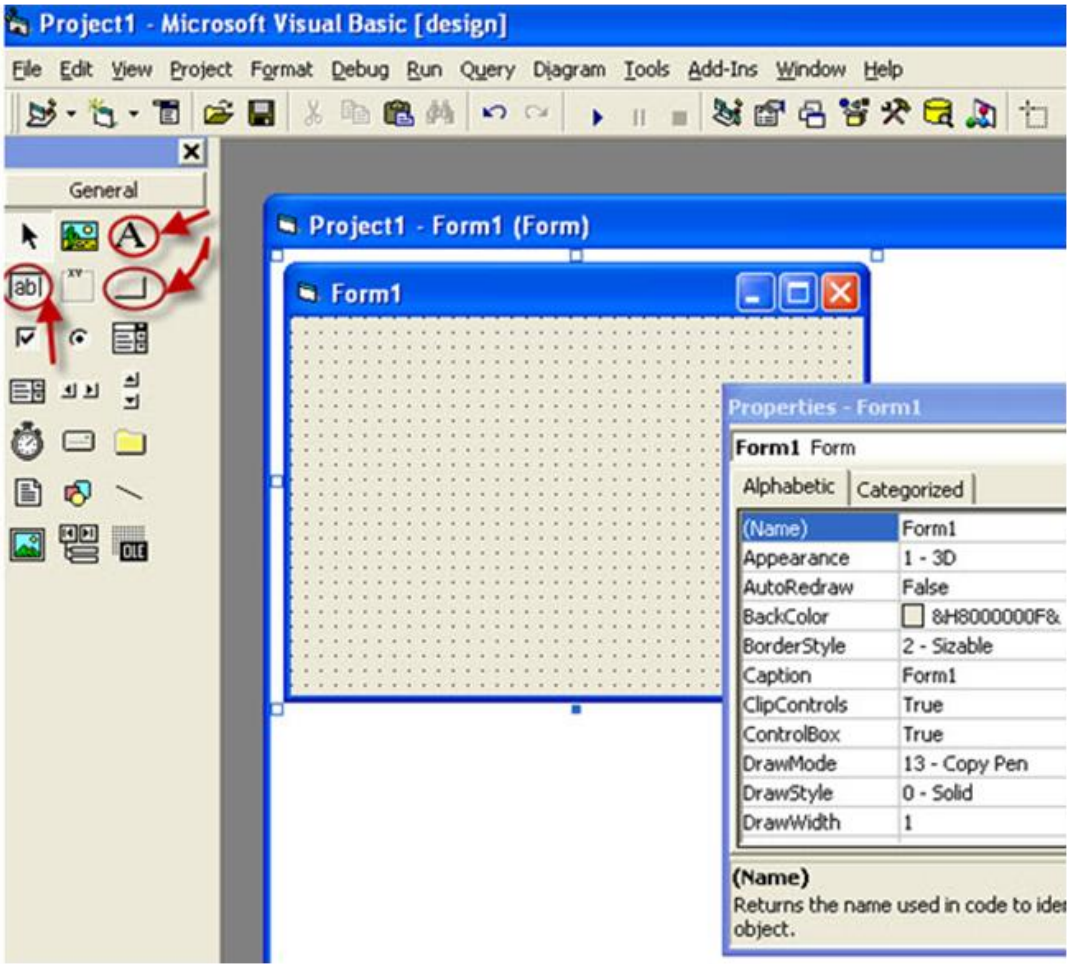


Figure 75. Adding Form in VB6 IDE.

The slider is not a common/basis object in visual basic environment. To add this object in the project toolbox, the developer must reference on additional library; "Microsoft Windows Common controls 6.0". As shown in the Figure 74, to achieve this task, the developer must click on the item "Component" of the project menu. In the list, he will check on the library and press the Apply command. Thus, the toolbox will be extended by new objects, including the Slider. To complete the user interface, the developer has to drag-and-drop an instance of the slider object.

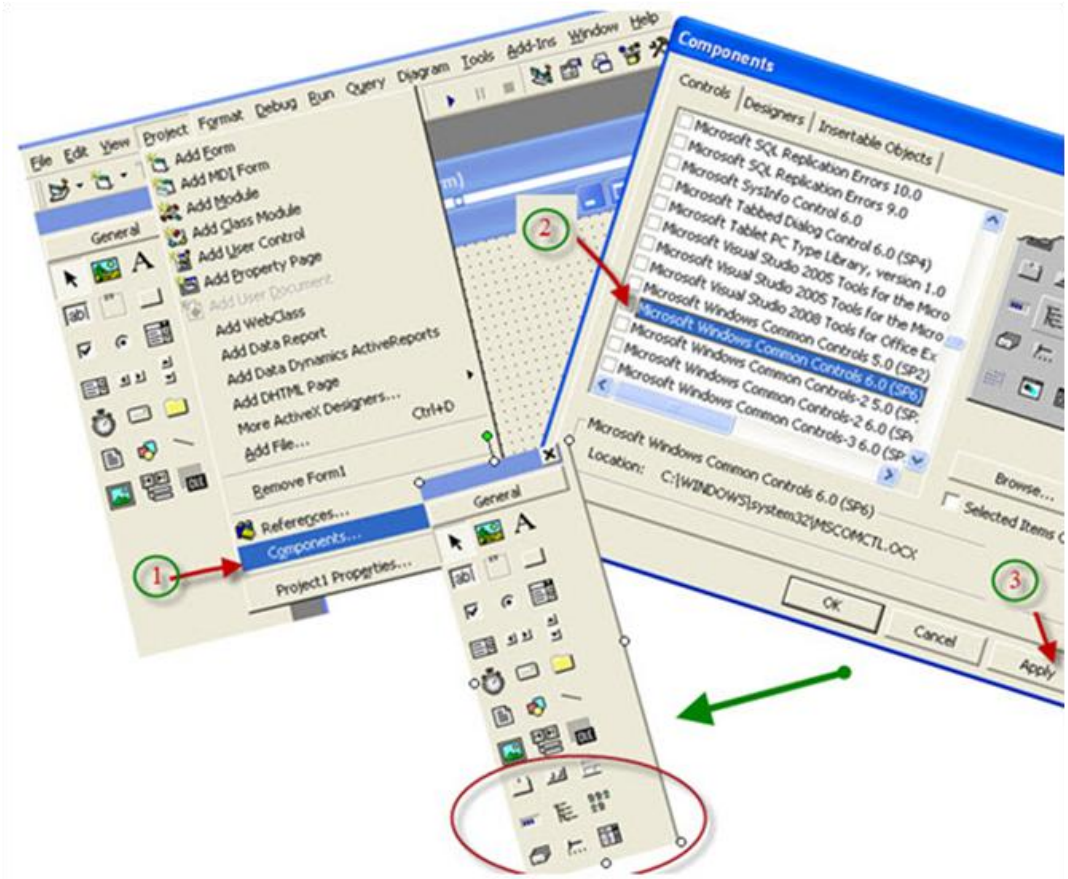


Figure 76. VB6 IDE, adding Component.

As shown in Figure 74, with Dialog Editor, a designer needs to open project manager tool. Afterwards, he must choose the specification level (Finalize) and a Toolkit (Visual basic 6.0). Thus, the list of available objects becomes accessible in the main table.

Here, developer inserts seven rows for a *Form*, two *command buttons*, two *labels*, a *textbox* and a *slider*. We must point out that there is no difference between common objects and additional objects. All objects are in the same list.
To resize objects and determine their locations, the developer must fill in the property values manually as shown in the Figure 75.

A.3.3 Semi-manual interface

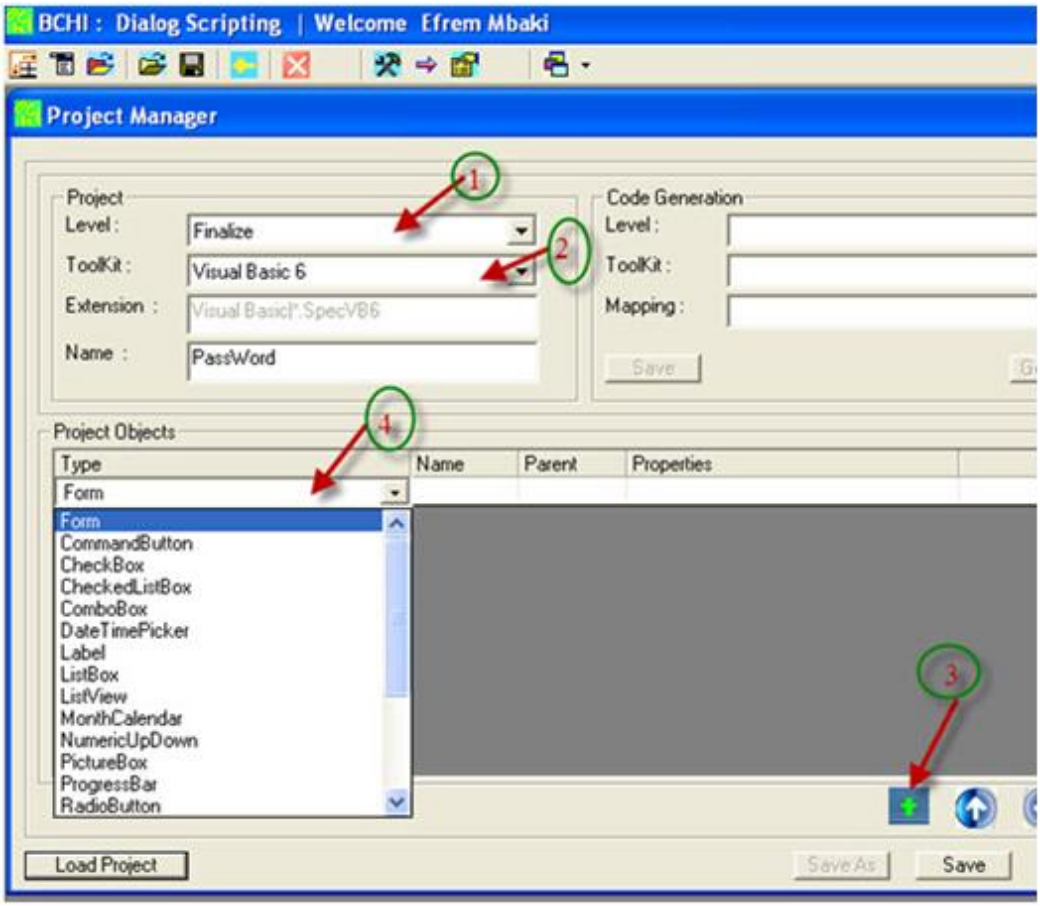


Figure 77. Dialog Editor, adding items.

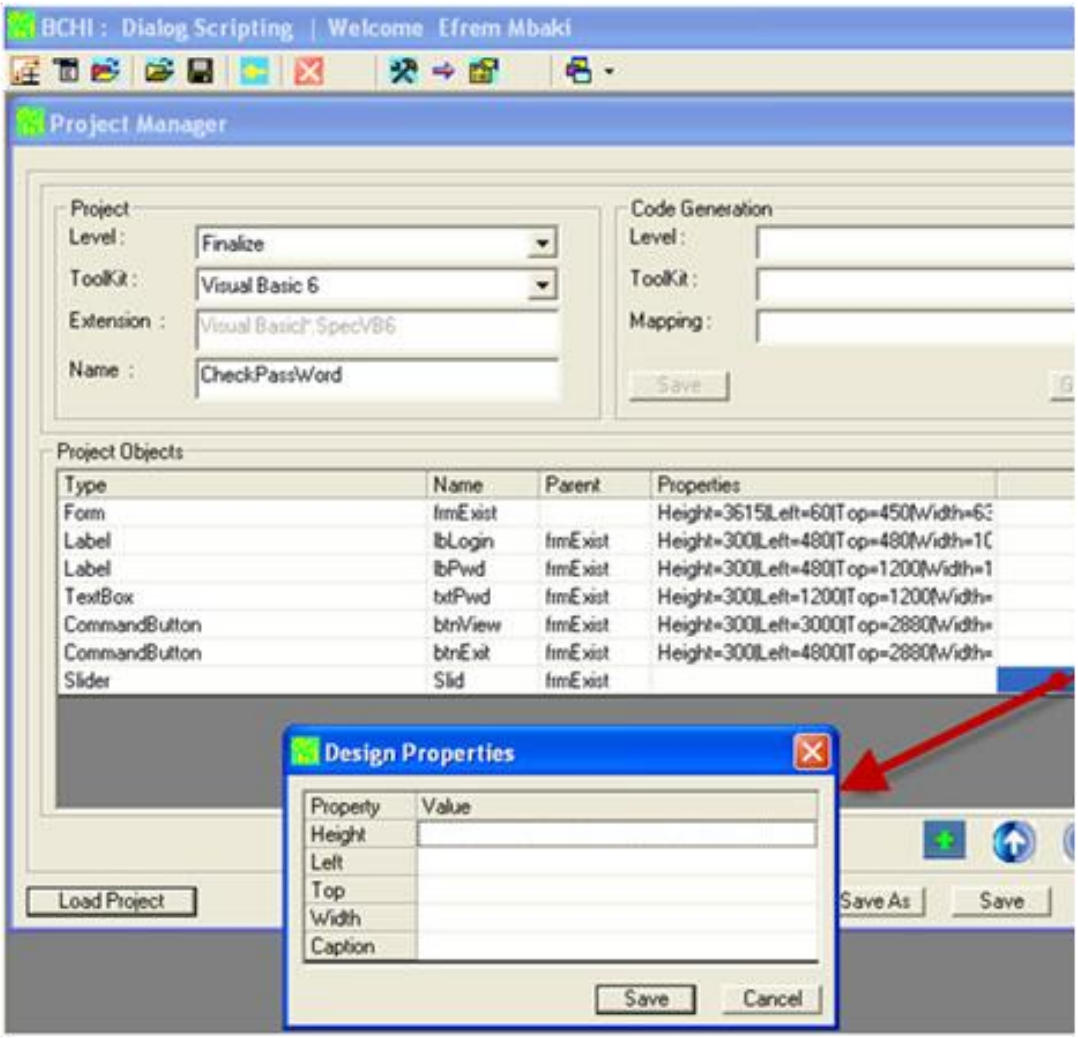


Figure 78. Dialog Editor, resizing item.

To the extent that he works at the final level, the developer has the choice of generating a code or changing the project in another *Toolkit*. In one click, the interface can be obtained in VBA, HTML or VB.Net. Similarly, without any programming knowledge and with a small command, a code for the specified interface can be generated. To achieve this task, he needs to fix the target level, the target toolkit and transforming mapping.

Annex A Password Evaluation

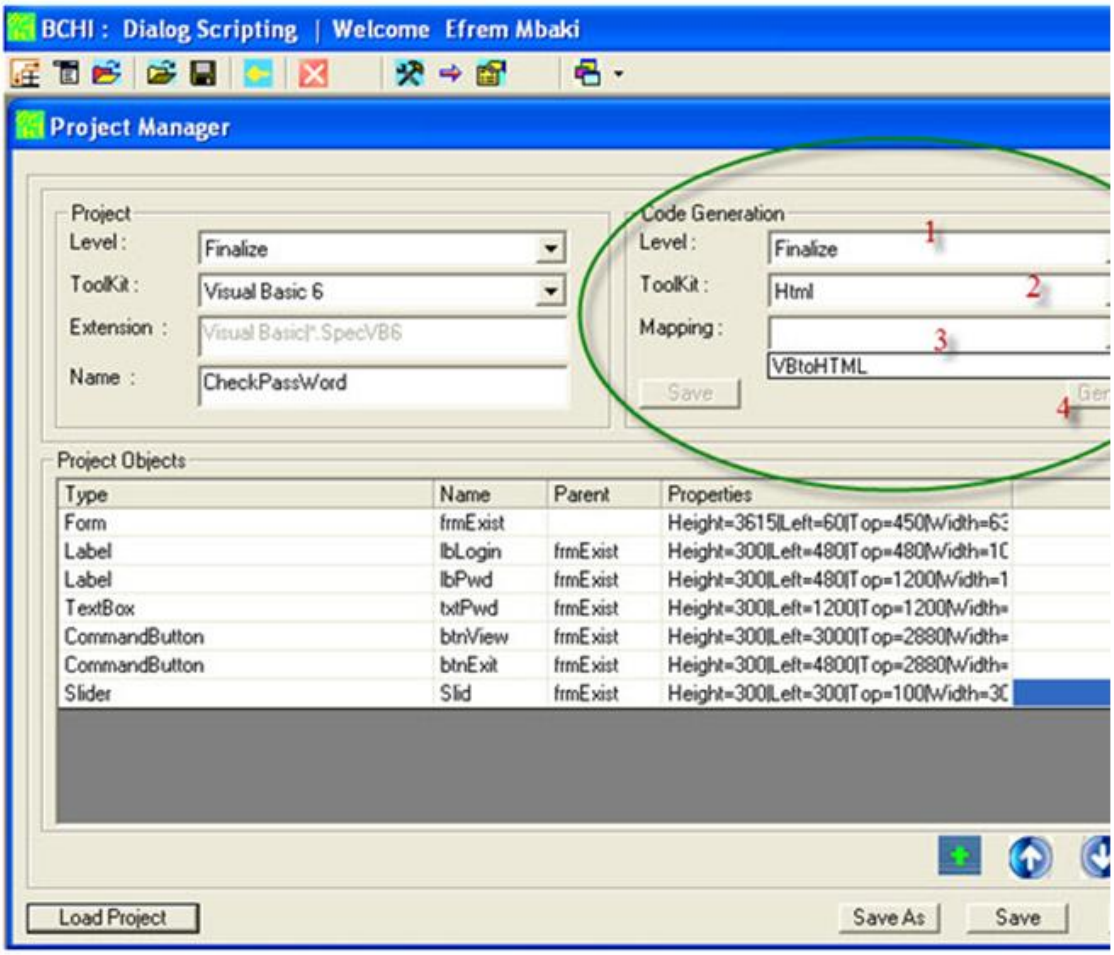


Figure 79. Dialog Editor, Choosing Mapping.

A.3.4. Comparison

Table 15: Comparison user interface

PROPERTY	MANUAL DESIGN	USING EDITOR
Base knowledge	Learn about Visual Basic Editor	Learn about Dialog Editor
Placing objects	By drag-and-drop, very easy to do	Filling geographic properties list, not easy to fix
Resizing objects	Using mouse and arrow keys, very easy to do	Filling size properties, not easy to imagine
Objects relationship	Visual, by drag-and-drop	By filling parent relationship using parent column
Complexity	Design form by form	Possibility of developing many forms at once.
Visualization	What you see is what you get	Without viewing the result interface, there exist the risk of having restart several times

Choice of Objects	Easy for common/basis objects. More complicated for advanced libraries objects	Easy to do; a simple list for all objects
Reuse	Almost impossible. Otherwise, create an ocx library to be used in another project that supports COM technology.	Several possible applications exploiting mappings. In addition, ability to switch to another level and / or toolkit without any line of code

A.4. Dialogue Programming

A.4.1. Dialogue Programming in visual basic

Although very simple, this example enables us to illustrate and implement certain very useful concepts. We can enumerate:

- *Functional Machine* which contains all semantics functions. Here, we use a class module named *cMachine* (Figure 79). The only function we need is the *Validate* function.

- The user interface is completely managed by a *specific controller*; the class module *cInterface* (Figure 80) captures events on the user interface and announces any exchange to the behaviour controller. Each event is characterized by three elements: its **source**, its **nature** and its **parameters**. It should be noticed that we do not mention the sender. Indeed, the same event can be captured by two or more senders.

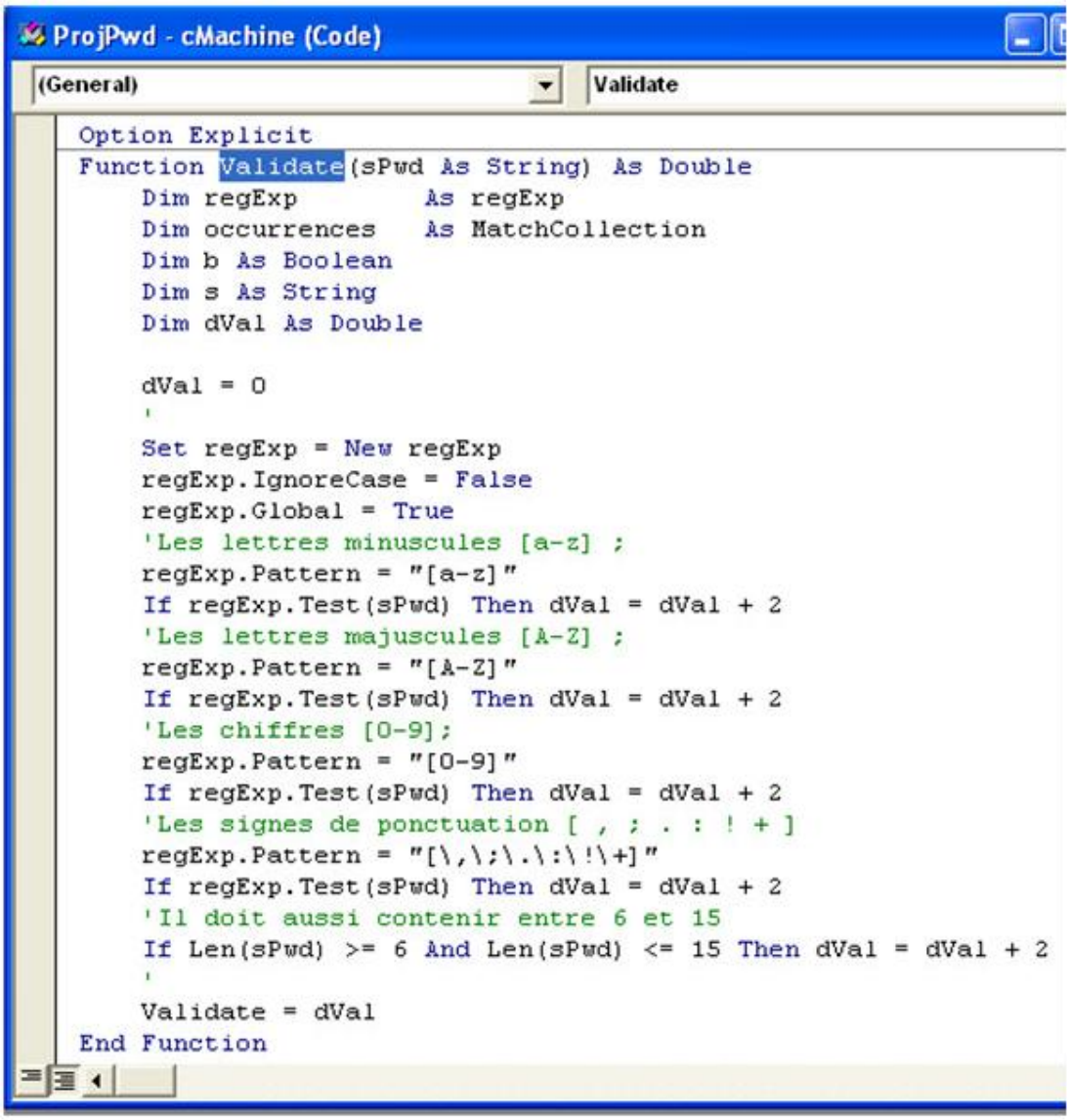


Figure 80. VB6 code of cMachine class.

ProjPwd - cInterface (Code)

(General) (Declarations)

```
Option Explicit
Private WithEvents PwdSlide As Slider: Private WithEvents cmdView   As CommandButton
Private WithEvents cmdClose As CommandButton: Private WithEvents txtPwd    As TextBox
Private frmMain             As frmMain: Private lbPerc              As Label
Public Event SendEvent(source As String, sEvent As String, cParam As Collection)
Sub setup(f As frmMain)
    Set frmMain = f
    With f: Set PwdSlide = .PwdSlider: Set cmdView = .cmdView
        Set txtPwd = .txtPwd: Set cmdClose = .cmdClose: Set lbPerc = .lbPerc: End With
    cmdView.Caption = "View"
    With PwdSlide: .Max = 10: .Min = 0: .Value = 0: End With
End Sub
Public Sub Display()
    frmMain.Show vbModal
End Sub
Private Sub cmdClose_Click()
    RaiseEvent SendEvent("cmdClose", "Click", Nothing)
End Sub
Private Sub cmdView_Click()
    RaiseEvent SendEvent("cmdView", "Click", Nothing)
End Sub

Sub ViewHideText(bView As Boolean)
    txtPwd.PasswordChar = IIf(bView, "*", "")
End Sub
Sub ViewHideLabel(bView As Boolean)
    cmdView.Caption = IIf(bView, "View", "Hide")
End Sub
Private Sub txtPwd_Change()
    Dim c As Collection:      Set c = New Collection:      c.Add txtPwd.Text, "text"
    RaiseEvent SendEvent("txtPwd", "Change", c)
End Sub
Sub SetValue(v As Double)
    PwdSlide.Value = v
    lbPerc.Caption = CStr(v * 10) & " %"
End Sub
```

Figure 81. VB6 code of Controller script.

- the use of the behaviour controller which manages exchanges between the functional machine and the user interface. We notice that this class uses two objects a controller of user interface and the functional machine. But also, two parameters;

    1. a Boolean (bView) which changes according to whether the password is visible or not. As we will see later, this Boolean represents two states of dialogue automat.

    2. a chain of the characters (sPwd) containing the current password running. To exploit an automat, we will need a pile to record the evolution of this word

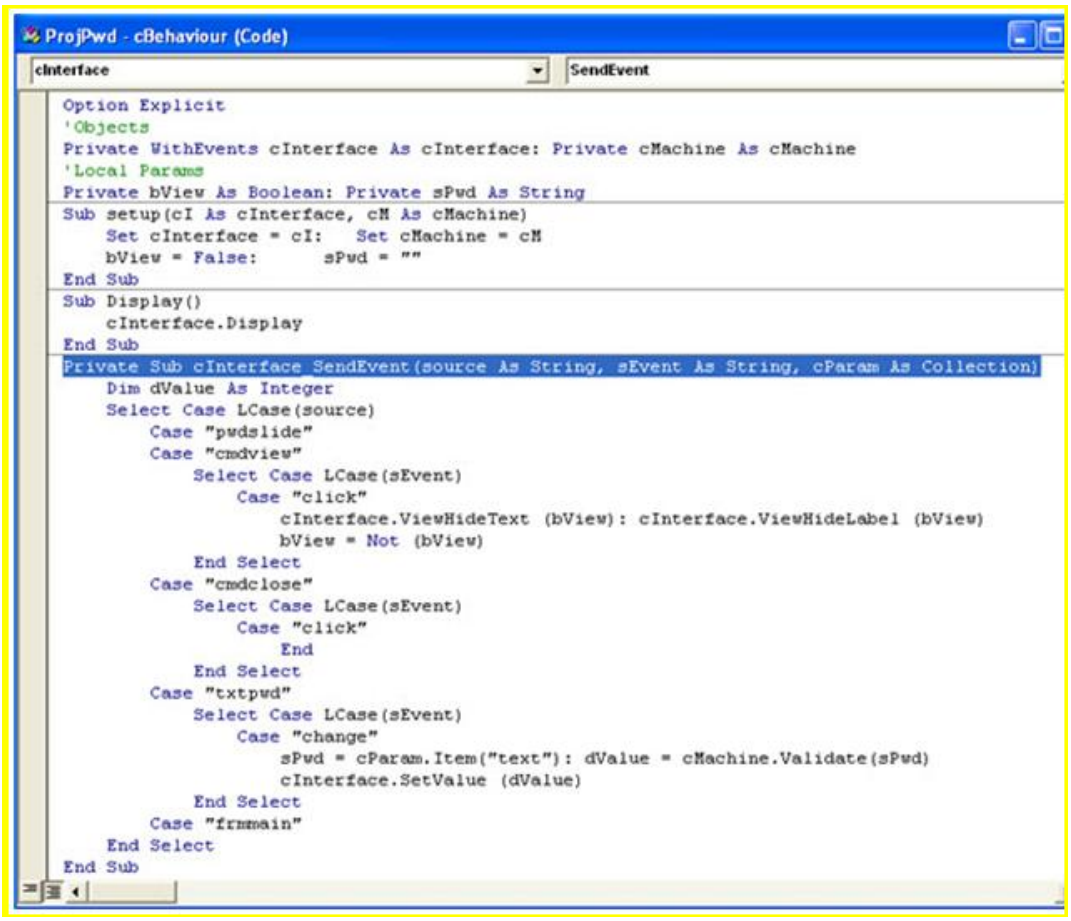


Figure 82. VB6 of cBehaviour class.

The initialization unit which prepares the necessary resources to start the application. In the project explorer below, this unit is implemented as a module named *mInit.bas*;

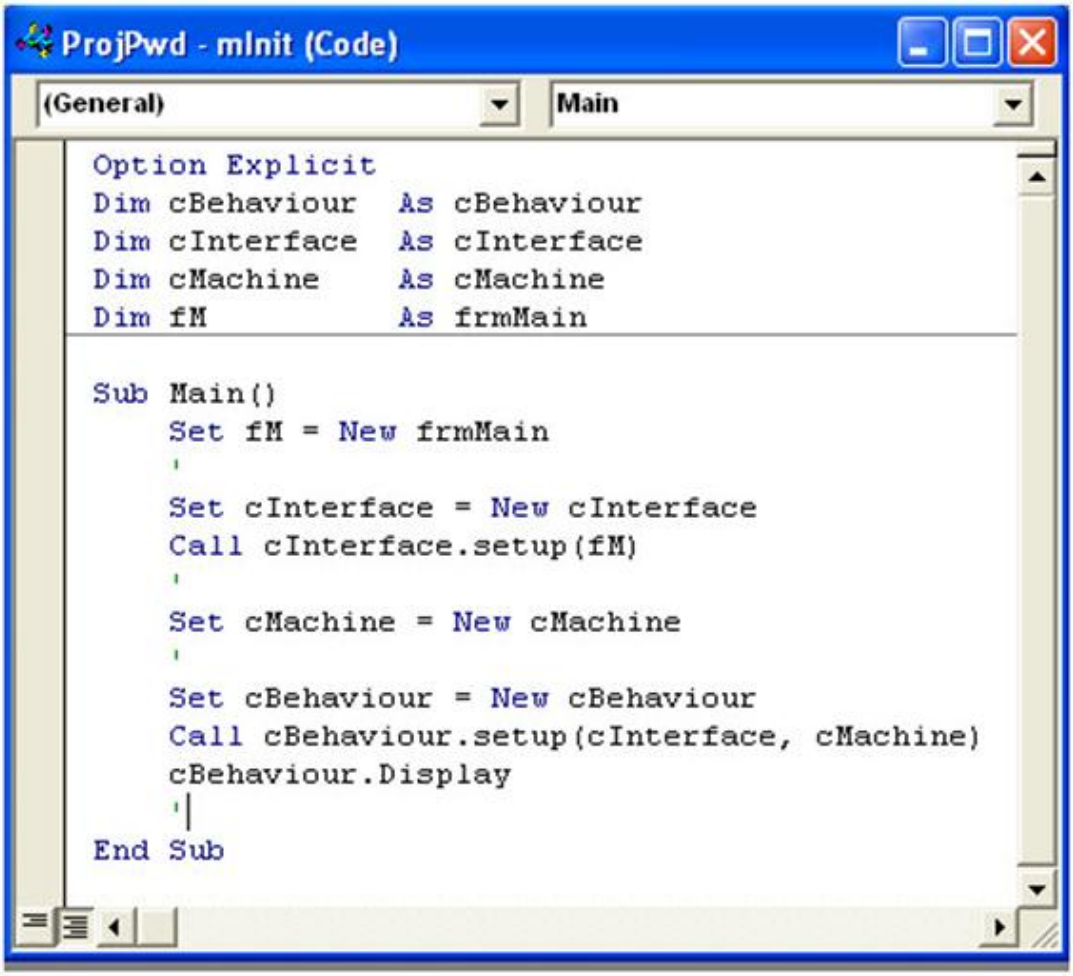


Figure 83. VB6 code of Initialization Module.

In summary, the visual basic project explorer below shows clearly how these components are integrated. We can list, one form, one module and three classes.

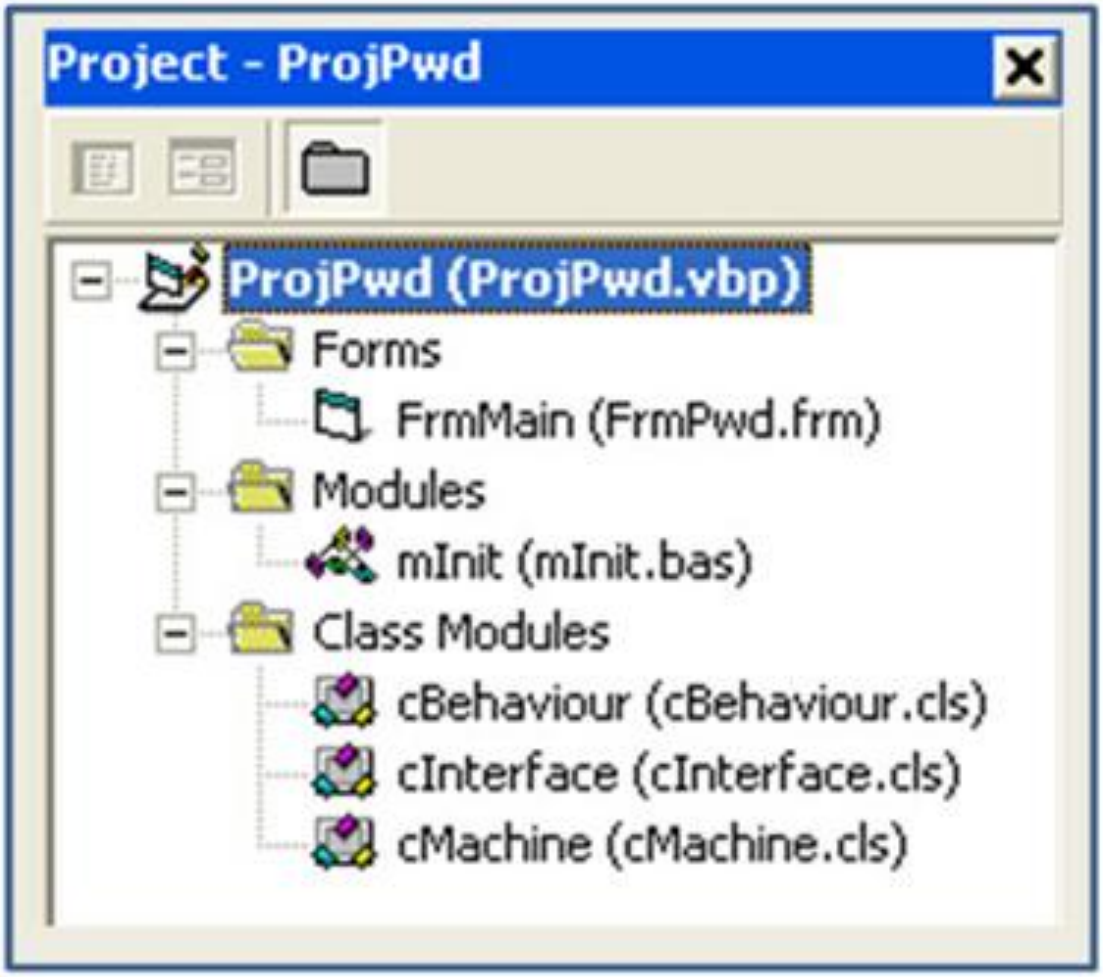


Figure 84. VB6 Project Explorer.

A.4.3. Dialogue specification with Dialog Editor

We hypothesize that the specified project is registered under vbp format i.e. as vb6 project. We want to open this project with the objective of specifying the dialogue. Let us open this project as shown in the Figure 83.

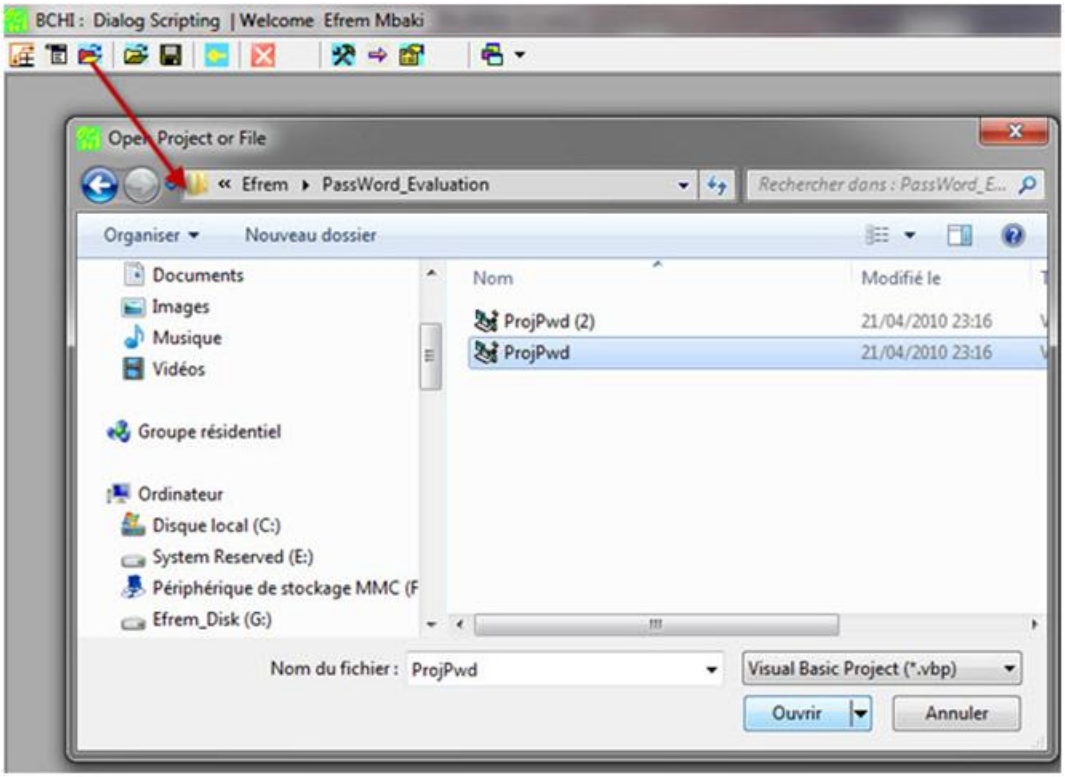


Figure 85. Opening Project.

As a result, the *Dialog Editor* presents a tree which lists all project components and/or objects. In addition, by clicking on each leaf of the tree, the editor offers a page where the developer can encode the script's dialogue node. Also, for some objects, there is a series of properties whose value choices can help to deduce certain characteristics of the dialogue. The following picture illustrates what we are saying.

Annex A Password Evaluation

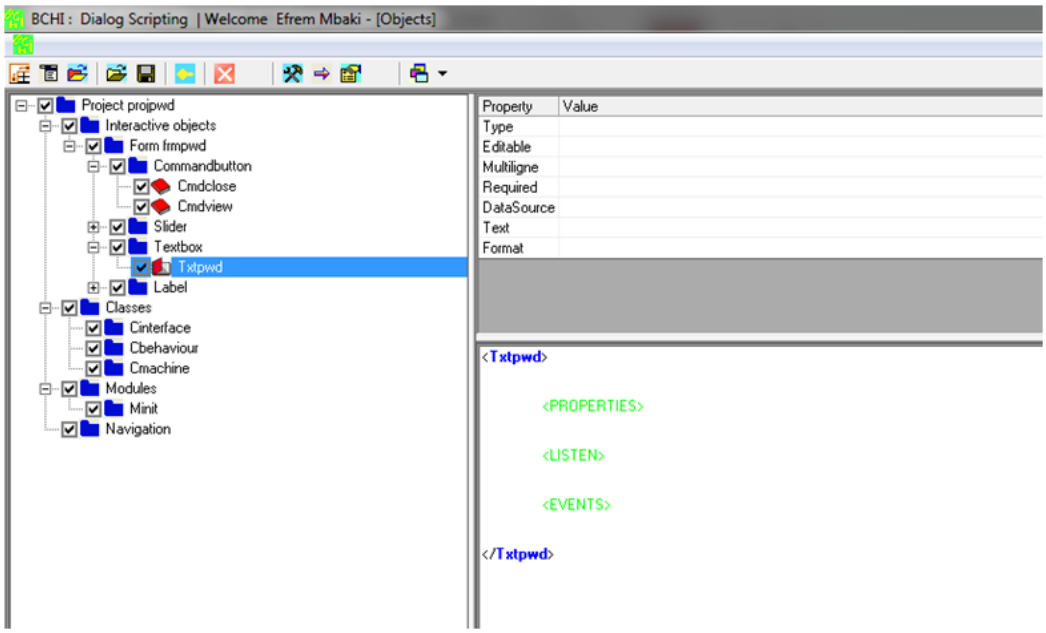


Figure 86. Project objects tree.

While programming the evaluation function requires several lines of code in the editor, this can be summarized in a few clicks as illustrated in Figure 86 below.

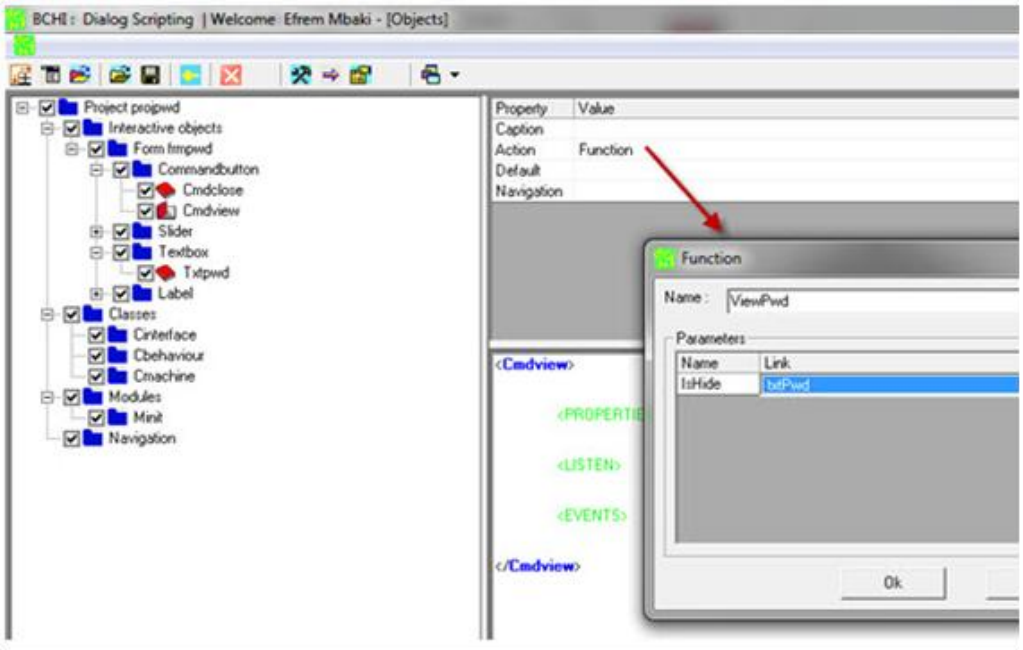


Figure 87. Fixing Properties.

In fact, we still have a few lines of code to specify the behaviour of the object. The big advantage here is that the script is written in a generic language. The best would be to increase the power of language tokens semantic in order to shorten these scripts. The

secret lies in the continuing effort to find a way to specify complex behaviours in two or three clicks.

Table 16. Comparison behaviour.

PROPERTY	MANUAL PROGRAMMING	USING EDITOR
Base knowledge	Visual Basic Programming	Learn about Dialog Editor
Event	Script to program fully	Specify using editor interface
Function	Script to program fully	Specify using editor interface; which parameter for which function; which parameter is attached to which interactive object
complexity	Complicated for complex function and/or Event script	Simplified by the power of semantic of language item.
Visualization	What you see is what you get	Without viewing of the result Interface, error-trying risk with many loop
Reuse	Almost impossible. Otherwise, create a DLL library to be used in another project that supports COM technology.	Several possible applications exploiting mappings. In addition, ability to switch to another level and / or toolkit without any line of code